ΧΕΙΡΙΣΜΟΣ ΑΙΤΗΣΕΩΝ ΠΕΛΑΤΗ: ΔΕΔΟΜΕΝΑ ΦΟΡΜΑΣ

Κεφάλαιο 4

Θέματα σε αυτό το Κεφάλαιο

- Ανάγνωση μεμονωμένων παραμέτρων αιτήσεων
- Ανάγνωση ολόκληρου του συνόλου των παραμέτρων αιτήσεων
- Χειρισμός ελλιπών δεδομένων ή δεδομένων με λανθασμένη μορφή
- Απομάκρυνση ειδικών χαρακτήρων από τις παραμέτρους αιτήσεων
- Αυτόματη συμπλήρωση ενός αντικειμένου δεδομένων με τιμές παραμέτρων αιτήσεων
- Χειρισμός των ημιτελών υποβολών φορμών

Ένα από τα βασικά κίνητρα για την κατασκευή δυναμικών ιστοσελίδων είναι ότι το αποτέλεσμα μπορεί να βασίζεται στην είσοδο του χρήστη. Σε αυτό το κεφάλαιο θα δούμε πώς μπορείτε να προσπελάσετε αυτή την είσοδο (Ενότητες 4.1–4.4). Θα δούμε επίσης πώς μπορείτε να χρησιμοποιείτε προεπιλεγμένες τιμές όταν λείπουν κάποιες από τις αναμενόμενες παραμέτρους (Ενότητα 4.5), πώς να απομακρύνετε τους χαρακτήρες < και > από τα δεδομένα αιτήσεων έτσι ώστε να αποφεύγετε το μπέρδεμα στα αποτελέσματα HTML (Ενότητα 4.6), πώς να δημιουργείτε "κόκκους φορμών" (form beans) οι οποίοι θα αναπαράγονται αυτόματα από τα δεδομένα αιτήσεων (Ενότητα 4.7), και πώς, όταν λείπουν κάποιες απαραίτητες παράμετροι από την αίτηση, να επαναεμφανίζετε τη φόρμα με επισημασμένες τις τιμές που λείπουν (Ενότητα 4.8).

4.1 Ο ρόλος των δεδομένων φόρμας

Αν έχετε χρησιμοποιήσει κάποια μηχανή αναζήτησης, αν έχετε επισκεφθεί ένα ηλεκτρονικό βιβλιοπωλείο, αν έχετε παρακολουθήσει μετοχές στον Ιστό, ή αν έχετε ρωτήσει κάποια τοποθεσία στον Ιστό για εκπτώσεις σε αεροπορικά εισιτήρια, τότε θα έχετε δει περίεργα URL της μορφής http://*υπολογιστήςΥπηρεσία*/*path*?user=Marty+Hall&origin=bwi&dest=sfo. Το τμήμα μετά το Αγγλικό ερωτηματικό (δηλαδή user=Marty+Hall&origin=bwi&dest=sfo) είναι γνωστό ως *δεδομένα φόρμας* (ή *δεδομένα ερωτήματος*) — form data ή query data — και είναι ο πιο συνηθισμένος τρόπος μεταφοράς πληροφοριών από μια ιστοσελίδα σε ένα πρόγραμμα το οποίο εκτελείται στο διακομιστή. Για τις αιτήσεις GET τα δεδομένα φόρμας μπορούν να προσαρτηθούν στο τέλος του URL μετά το ερωτηματικό (όπως παραπάνω)· για αιτήσεις POST τα δεδομένα φόρμας μπορούν επίσης να αποστάλούν στο διακομιστή σε ξεχωριστή γραμμή. Αν δεν είστε εξοικειωμένοι με τις φόρμες

HTML, στο Κεφάλαιο 19 (Δημιουργία και επεξεργασία φορμών HTML) θα βρείτε πληροφορίες για το πώς μπορείτε να κατασκευάσετε φόρμες που θα συγκεντρώνουν και θα μεταδίδουν δεδομένα αυτού του τύπου· εδώ θα δούμε απλώς τα βασικά στοιχεία.

1. **Χρήση του στοιχείου FORM για τη δημιουργία μιας φόρμας HTML.** Χρησιμοποιήστε την ιδιότητα ACTION για να ορίσετε τη διεύθυνση της μικροϋπηρεσίας ή της σελίδας JSP που θα επεξεργαστεί τα αποτελέσματα· μπορείτε να χρησιμοποιήσετε είτε απόλυτη είτε σχετική διεύθυνση URL. Για παράδειγμα:

   ```
   <FORM ACTION="...">...</FORM>
   ```

   Αν παραλείψετε την ιδιότητα ACTION, τα δεδομένα θα υποβληθούν στο URL της τρέχουσας σελίδας.

2. **Χρήση στοιχείων εισόδου για τη συλλογή των δεδομένων του χρήστη.** Τοποθετήστε τα στοιχεία μεταξύ των ετικετών αρχής και τέλους του στοιχείου FORM και αποδώστε τιμή στην ιδιότητα NAME του κάθε στοιχείου εισόδου. Τα πεδία κειμένου είναι το πιο συνηθισμένο στοιχείο εισόδου· η δημιουργία τους γίνεται με την εντολή

   ```
   <INPUT TYPE="TEXT" NAME="...">
   ```

3. **Τοποθέτηση ενός κουμπιού υποβολής (submit) κοντά στο τέλος της φόρμας.** Για παράδειγμα:

   ```
   <INPUT TYPE="SUBMIT">
   ```

   Όταν ο χρήστης πατά στο κουμπί, καλείται το URL που έχει οριστεί στην ιδιότητά ACTION της φόρμας. Στις αιτήσεις GET στο τέλος του URL προσαρτάται ένα ερωτηματικό και ζεύγη ονομάτων/τιμών, όπου τα ονόματα προέρχονται από τις ιδιότητες NAME των στοιχείων εισόδου HTML και οι τιμές προέρχονται από τον τελικό χρήστη. Στις αιτήσεις POST αποστέλλονται τα ίδια ακριβώς δεδομένα, αλλά τοποθετούνται σε μία ξεχωριστή γραμμή αίτησης και δεν προσαρτώνται στο URL.

Η εξαγωγή των απαραίτητων πληροφοριών από αυτά τα δεδομένα φορμών αποτελεί παραδοσιακά μια από τις πλέον επίπονες διαδικασίες του προγραμματισμού εφαρμογών που εκτελούνται στο διακομιστή.

Καταρχήν, πριν από την έλευση των μικροϋπηρεσιών, έπρεπε να διαβάσετε με κάποιον τρόπο τα δεδομένα των αιτήσεων GET (στις παραδοσιακές σελίδες CGI, αυτό γίνεται συνήθως με τη μεταβλητή περιβάλλοντος QUERY_STRING) και με διαφορετικό τρόπο τα δεδομένα των αιτήσεων POST (στις σελίδες CGI αυτό γίνεται με ανάγνωση της τυπικής εισόδου).

Δεύτερον, θα πρέπει να αποκόψετε τα ζεύγη εκεί όπου υπάρχουν σύμβολα εμπορικού και (&) και κατόπιν να διαχωρίσετε τα ονόματα των παραμέτρων (στα αριστερά του συμβόλου ίσον) από τις τιμές των παραμέτρων (στα δεξιά του συμβόλου ίσον).

Τρίτον, θα πρέπει να *αποκωδικοποιήσετε τις τιμές του URL* (URL-decode): δηλαδή να "αντιστρέψετε" την κωδικοποίηση που χρησιμοποιεί ο φυλλομετρητής για ορισμένους χαρακτήρες. Οι αλφαριθμητικοί χαρακτήρες αποστέλλονται από το φυλλομετρητή ως έχουν, αλλά τα διαστήματα μετατρέπονται σε σύμβολα συν (+) και κάποιοι άλλοι χαρακτήρες μετατρέπονται σε *%XX*, όπου *XX* είναι η τιμή ASCII (ή ISO Latin-1) του χαρακτήρα, στο δεκαεξαδικό σύστημα. Για παράδειγμα, αν κάποιος εισαγάγει την τιμή "~hall, ~gates, and ~mcnealy" σε ένα πεδίο κειμένου με το όνομα users μιας φόρμας HTML, τα δεδομένα θα σταλούν ως "users=%7Ehall%2C+%7Egates%2C+and+%7Emcnealy", και το πρόγραμμα στην πλευρά του διακομιστή θα πρέπει να επανασυνθέσει το αρχικό αλφαριθμητικό.

Τέλος, ο τέταρτος λόγος για τον οποίο είναι επίπονη η συντακτική ανάλυση δεδομένων φορμών με τις παραδοσιακές τεχνικές προγραμματισμού στο διακομιστή είναι ότι μπορούν να παραλειφθούν τιμές (π.χ. `param1=val&param2=&param3=val3`) ή η παράμετρος μπορεί να εμφανίζεται περισσότερες από μία φορές (π.χ. `param1=val&param2=val2&param1=val3`), οπότε ο κώδικας συντακτικής ανάλυσης χρειάζεται ειδικό χειρισμό για αυτές τις καταστάσεις.

Ευτυχώς, οι μικροϋπηρεσίες μάς βοηθούν πολύ σε αυτή την επίπονη διαδικασία της συντακτικής ανάλυσης. Αυτό ακριβώς είναι το θέμα της επόμενης ενότητας.

4.2 Ανάγνωση δεδομένων φορμών από μικροϋπηρεσίες

Ένα από τα ωραία χαρακτηριστικά των μικροϋπηρεσιών είναι ότι όλη αυτή η συντακτική ανάλυση φορμών γίνεται αυτόματα. Καλείτε απλώς τη `request.getParameter` για να πάρετε την τιμή μιας παραμέτρου φόρμας. Μπορείτε επίσης να καλέσετε τη `request.getParameterValues` αν η παράμετρος εμφανίζεται περισσότερες από μία φορές, ή μπορείτε να καλέσετε τη `request.getParameterNames` αν θέλετε να πάρετε έναν πλήρη κατάλογο όλων των παραμέτρων της τρέχουσας αίτησης. Στις σπάνιες περιπτώσεις που θέλετε να διαβάσετε τα ανεπεξέργαστα δεδομένα της αίτησης έτσι ώστε να κάνετε εσείς οι ίδιοι τη συντακτική ανάλυση, καλείτε την `getReader` ή την `getInputStream`.

Ανάγνωση μιας τιμής: getParameter

Για να διαβάσετε μια παράμετρο αίτησης (φόρμας), καλείτε απλώς τη μέθοδο `getParameter` της κλάσης `HttpServlet`, δίνοντας ως όρισμα το όνομα της παραμέτρου (με διάκριση πεζών-κεφαλαίων). Το όνομα της παραμέτρου που δίνετε πρέπει να είναι ακριβώς το ίδιο με εκείνο του πηγαίου κώδικα HTML, και το αποτέλεσμα που παίρνετε είναι ακριβώς όπως το εισήγαγε ο τελικός χρήστης· αν είναι απαραίτητη η αποκωδικοποίηση, αυτή γίνεται αυτόματα. Σε αντίθεση με άλλες, εναλλακτικές ως προς τις μικροϋπηρεσίες τεχνολογίες, χρησιμοποιείτε την `getParameter` με τον ίδιο τρόπο είτε τα δεδομένα αποστέλλονται με αίτηση GET (δηλαδή από μια `doGet`) είτε αποστέλλονται με αίτηση POST (δηλαδή από μια `doPost`)· η μικροϋπηρεσία γνωρίζει ποια μέθοδο αίτησης χρησιμοποίησε ο πελάτης και χρησιμοποιεί αυτόματα την κατάλληλη μέθοδο για να διαβάσει τα δεδομένα. Αν η παράμετρος υπάρχει αλλά δεν έχει τιμή (δηλαδή ο χρήστης άφησε κενό το αντίστοιχο πεδίο κειμένου κατά την υποβολή της φόρμας) επιστρέφεται ένα κενό αντικείμενο `String`, ενώ αν δεν υπάρχει τέτοια παράμετρος επιστρέφεται τιμή `null`.

Στα ονόματα των παραμέτρων γίνεται διάκριση πεζών-κεφαλαίων. Για παράδειγμα, τα `request.getParameter("Param1")` και `request.getParameter("param1")` *δεν* είναι ίδια.

Προειδοποίηση

Στις τιμές που δίνονται στις μεθόδους getParameter και getParameterValue γίνεται διάκριση πεζών-κεφαλαίων.

Ανάγνωση πολλών τιμών: getParameterValues

Αν το ίδιο όνομα παραμέτρου μπορεί να εμφανιστεί περισσότερες από μία φορές στα δεδομένα της φόρμας, θα πρέπει να καλέσετε τη μέθοδο `getParameterValues` (η οποία επιστρέφει έναν πίνακα αλφαριθμητικών) αντί για τη μέθοδο `getParameter` (η οποία επιστρέφει ένα αλφαριθμητικό που αντιστοιχεί στην πρώτη παρουσία της παραμέτρου). Στην περίπτωση που η παράμετρος δεν υπάρχει η επιστρεφόμενη τιμή της `getParameterValues` είναι `null`, ενώ όταν η παράμετρος έχει μία μόνο τιμή τότε η επιστρεφόμενη τιμή είναι ένας πίνακας με ένα μόνο στοιχείο.

Βέβαια, αν είστε εσείς οι δημιουργοί της φόρμας HTML, είναι συνήθως καλύτερο να εξασφαλίσετε ότι κάθε πεδίο κειμένου, πλαίσιο ελέγχου (checkbox), ή άλλο στοιχείο της διασύνδεσης χρήστη έχει μοναδικό όνομα. Έτσι θα μπορείτε να χρησιμοποιήσετε την απλούστερη μέθοδο `getParameter` και να αποφύγετε εντελώς την `getParameterValues`. Μερικές φορές, όμως, θα γράφετε μικροϋπηρεσίες και σελίδες JSP που θα χειρίζονται φόρμες HTML τις οποίες έχουν δημιουργήσει άλλοι, οπότε θα πρέπει να είστε σε θέση να χειριστείτε όλες τις πιθανές περιπτώσεις. Εκτός απ' αυτό, τα πλαίσια καταλόγου πολλαπλής επιλογής (δηλαδή, τα στοιχεία `SELECT` της HTML όπου είναι ενεργοποιημένη η ιδιότητα `MULTIPLE` — δείτε το Κεφάλαιο 19) επαναλαμβάνουν το όνομα της παραμέτρου για κάθε επιλεγμένο στοιχείο του καταλόγου. Έτσι, δεν μπορείτε πάντοτε να αποφύγετε τις πολλαπλές τιμές.

Αναζήτηση ονομάτων παραμέτρων: getParameterNames και getParameterMap

Οι πιο πολλές μικροϋπηρεσίες αναζητούν ένα συγκεκριμένο σύνολο ονομάτων παραμέτρων· στις περισσότερες περιπτώσεις, αν η μικροϋπηρεσία δεν γνωρίζει το όνομα της παραμέτρου, τότε δεν γνωρίζει και τι να κάνει με αυτή. Έτσι το βασικό σας εργαλείο θα πρέπει να είναι η μέθοδος `getParameter`. Μερικές φορές, όμως, είναι χρήσιμο να έχετε μια πλήρη λίστα με τα ονόματα των παραμέτρων. Ο κύριος σκοπός της πλήρους λίστας είναι η αποσφαλμάτωση, αλλά περιστασιακά θα χρησιμοποιείτε τον κατάλογο για εφαρμογές όπου τα ονόματα παραμέτρων είναι πολύ δυναμικά. Για παράδειγμα, τα ίδια τα ονόματα μπορεί να λένε στο σύστημα τι να κάνει με τις παραμέτρους (π.χ. `row-1-col-3-value`, δηλαδή "τιμή στη γραμμή 1 και στήλη 3"), ή μπορεί το σύστημα να πραγματοποιεί μια ενημέρωση βάσης δεδομένων με την προϋπόθεση ότι τα ονόματα παραμέτρων είναι τα ονόματα των στηλών της βάσης δεδομένων, ή η μικροϋπηρεσία μπορεί να αναζητά κάποια συγκεκριμένα ονόματα και τα υπόλοιπα να τα μεταβιβάζει σε κάποια άλλη εφαρμογή.

Με τη μέθοδο `getParameterNames` μπορείτε να πάρετε αυτή τη λίστα με τη μορφή απαρίθμησης (`Enumeration`), όπου κάθε καταχώριση μπορεί να μετατραπεί ρητά (cast) σε τύπο δεδομένων `String` και να χρησιμοποιηθεί σε μια κλήση `getParameter` ή `getParameterValues`. Αν στην τρέχουσα αίτηση δεν υπάρχουν παράμετροι, τότε η `getParameterNames` επιστρέφει μια κενή απαρίθμηση (και όχι μια τιμή `null`). Ας σημειωθεί ότι η απαρίθμηση (`Enumeration`) είναι μια διασύνδεση που εγγυάται απλώς ότι η πραγματική κλάση θα διαθέτει τις μεθόδους `hasMoreElements` και `nextElement`: δεν υπάρχει κάποια εγγύηση ότι θα χρησιμοποιηθεί κάποια συγκεκριμένη υποκείμενη δομή δεδομένων. Επειδή ορισμένες συνηθισμένες δομές δεδομένων (ιδιαίτερα οι πίνακες κατακερματισμού, hash tables) ανακατεύουν τη σειρά των στοιχείων, δεν θα πρέπει να θεωρείτε ότι η μέθοδος `getParameterNames` θα επιστρέψει τις παραμέτρους με τη σειρά που εμφανίζονται στη φόρμα HTML.

Προειδοποίηση

Μη βασίζεστε στο ότι η μέθοδος `getParameterNames` θα επιστρέψει τα ονόματα με κάποια συγκεκριμένη σειρά.

Μια εναλλακτική λύση ως προς τη μέθοδο `getParameterNames` είναι η μέθοδος `getParameterMap`. Αυτή η μέθοδος επιστρέφει ένα χάρτη (`Map`): τα ονόματα των παραμέτρων (αλφαριθμητικά) είναι τα κλειδιά του πίνακα και οι τιμές των παραμέτρων (πίνακες αλφαριθμητικών όπως επιστρέφονται από την `getParameterNames`) είναι οι τιμές του πίνακα.

Ανάγνωση ανεπεξέργαστων δεδομένων φορμών και συντακτική ανάλυση αρχείων που φορτώνονται στο διακομιστή: getReader και getInputStream

Αντί να διαβάσετε μεμονωμένες παραμέτρους φόρμας, μπορείτε να προσπελάσετε άμεσα τα δεδομένα ερωτήματος καλώντας τη μέθοδο `getReader` ή τη μέθοδο `getInputStream` για το αντικείμενο της κλάσης `HttpServletRequest`, και μετά να χρησιμοποιήσετε αυτό το ρεύμα (stream) για να αναλύσετε συντακτικά τα ανεπεξέργαστα δεδομένα της εισόδου. Θα πρέπει να σημειώσουμε όμως ότι, αν διαβάσετε τα δεδομένα με αυτόν τον τρόπο, δεν υπάρχει εγγύηση ότι θα είναι διαθέσιμα με τη μέθοδο `getParameter`.

Η ανάγνωση των ανεπεξέργαστων δεδομένων είναι καινή ιδέα όσον αφορά τις κανονικές παραμέτρους αφού η είσοδος ούτε αναλύεται συντακτικά (δεν διαχωρίζεται σε καταχωρίσεις ειδικά για κάθε παράμετρο), ούτε γίνεται αποκωδικοποίηση URL (δηλαδή δεν μεταφράζεται έτσι ώστε τα σύμβολα συν να γίνουν διαστήματα και τα σύμβολα *%XX* να αντικατασταθούν από τους αρχικούς χαρακτήρες ASCII ή ISO Latin-1 που αντιστοιχούν στη δεκαεξαδική τιμή *XX*). Ωστόσο, η ανάγνωση της ανεπεξέργαστης εισόδου είναι χρήσιμη σε δύο περιπτώσεις.

Η πρώτη περίπτωση στην οποία μπορεί να θέλετε να διαβάσετε και να αναλύσετε συντακτικά τα δεδομένα είναι όταν τα δεδομένα προέρχονται από έναν προσαρμοσμένο πελάτη, και όχι από κάποια φόρμα HTML. Η πιο συνηθισμένη περίπτωση προσαρμοσμένων πελατών είναι οι μικροεφαρμογές (applets): η επικοινωνία μικροεφαρμογών-μικροϋπηρεσιών εξετάζεται στο βιβλίο More Servlets and JavaServer Pages.

Η δεύτερη περίπτωση στην οποία μπορεί να θέλετε να διαβάσετε εσείς τα δεδομένα είναι όταν τα δεδομένα προέρχονται από κάποιο αρχείο που έχει φορτωθεί στο διακομιστή (uploaded file). Η HTML υποστηρίζει ένα στοιχείο FORM (<INPUT TYPE="FILE"...) που επιτρέπει στον πελάτη να φορτώσει ένα αρχείο στο διακομιστή. Δυστυχώς, η διασύνδεση των μικροϋπηρεσιών δεν ορίζει κάποιο μηχανισμό ανάγνωσης τέτοιων αρχείων. Έτσι θα χρειαστείτε κάποια βιβλιοθήκη τρίτου κατασκευαστή για να το κάνετε αυτό. Μια από τις πιο δημοφιλείς βιβλιοθήκες προέρχεται από την ομάδα Apache Jakarta Project. Για λεπτομέρειες, δείτε τη διεύθυνση http://Jakarta.apache.org/commons/fileupload/.

Ανάγνωση εισόδου για περισσότερα από ένα σύνολα χαρακτήρων: setCharacterEncoding

Εξ ορισμού, η μέθοδος request.getParameter ερμηνεύει την είσοδο με βάση το τρέχον σύνολο χαρακτήρων του διακομιστή. Για να αλλάξετε αυτή την προεπιλεγμένη συμπεριφορά, χρησιμοποιήστε τη μέθοδο setCharacterEncoding της κλάσης ServletRequest. Τι θα συμβεί, όμως, αν η είσοδος μπορεί να ανήκει σε περισσότερα από ένα σύνολα χαρακτήρων; Σε μια τέτοια περίπτωση δεν μπορείτε να καλέσετε απλώς τη μέθοδο setCharacterEncoding με ένα κανονικό όνομα συνόλου χαρακτήρων. Ο λόγος γι' αυτόν τον περιορισμό οφείλεται στο ότι θα πρέπει η setCharacterEncoding να κληθεί *πριν* προσπελάσετε τις παραμέτρους της αίτησης, και σε πολλές περιπτώσεις θα χρησιμοποιείτε μια παράμετρο της αίτησης (για παράδειγμα, ένα πλαίσιο ελέγχου) προκειμένου να καθορίσετε το σύνολο χαρακτήρων.

Έτσι σας απομένουν δύο επιλογές: να διαβάσετε την παράμετρο σε ένα σύνολο χαρακτήρων και μετά να μετατρέψετε το κείμενο σε ένα άλλο σύνολο χαρακτήρων, ή να χρησιμοποιήσετε τη λειτουργία αυτόματης ανίχνευσης που παρέχεται από ορισμένα σύνολα χαρακτήρων.

Όσον αφορά την πρώτη επιλογή, θα διαβάσετε την παράμετρο που σας ενδιαφέρει, θα χρησιμοποιήσετε τη μέθοδο getBytes για να εξαγάγετε τα ανεπεξέργαστα byte, και μετά θα μεταβιβάσετε αυτά τα byte στη συνάρτηση δόμησης (constructor) ενός String μαζί με το όνομα του επιθυμητού συνόλου χαρακτήρων. Το ακόλουθο είναι ένα παράδειγμα που μετατρέπει μια παράμετρο στα Ιαπωνικά:

```
String firstNameWrongEncoding = request.getParameter("firstName");
String firstName =
  new String(firstNameWrongEncoding.getBytes(), "Shift_JIS");
```

Για τη δεύτερη επιλογή θα χρησιμοποιήσετε ένα σύνολο χαρακτήρων που υποστηρίζει την ανίχνευση και μετατροπή από το προεπιλεγμένο σύνολο χαρακτήρων. Στην ιστοσελίδα http://java.sun.com/j2se/1.4.1/docs/guide/intl/encoding.doc.html μπορείτε να δείτε έναν πλήρη κατάλογο των συνόλων χαρακτήρων που υποστηρίζονται από τη Java. Για παράδειγμα, για να επιτρέψετε την είσοδο είτε στα Αγγλικά είτε στα Ιαπωνικά, μπορείτε να χρησιμοποιήσετε τον ακόλουθο κώδικα:

```
request.setCharacterEncoding("JISAutoDetect");
String firstName = request.getParameter("firstName");
```


4.3 Παράδειγμα: Ανάγνωση τριών παραμέτρων

Η Λίστα 4.1 παρουσιάζει μια απλή μικροϋπηρεσία με όνομα ThreeParams, η οποία διαβάζει τις παραμέτρους φόρμας param1, param2, και param3 και τοποθετεί τις τιμές τους σε μια λίστα με κουκκίδες. Αν και είστε υποχρεωμένοι να καθορίσετε τις ρυθμίσεις *απάντησης* (δείτε τα Κεφάλαια 6 και 7) πριν αρχίσετε να παράγετε το περιεχόμενο, δεν είστε υποχρεωμένοι να διαβάσετε τις παραμέτρους *αίτησης* σε κάποια συγκεκριμένη θέση του κώδικά σας. Έτσι, εδώ θα διαβάσουμε τις παραμέτρους μόνο όταν είμαστε έτοιμοι να τις χρησιμοποιήσουμε. Θα πρέπει επίσης να θυμάστε ότι, αφού η κλάση ThreeParams βρίσκεται στο πακέτο coreservlets, η εκδίπλωση της μικροϋπηρεσίας θα γίνει στον υποκατάλογο coreservlets του καταλόγου WEB-INF/classes της εφαρμογής Ιστού (στη συγκεκριμένη περίπτωση, της προεπιλεγμένης εφαρμογής Ιστού).

Όπως θα δούμε αργότερα, αυτή η μικροϋπηρεσία αποτελεί εξαιρετικό παράδειγμα περίπτωσης που είναι εξαιρετικά απλούστερη με σελίδα JSP. Στην Ενότητα 11.6 (Σύγκριση μικροϋπηρεσιών και σελίδων JSP) μπορείτε να δείτε την ισοδύναμη εκδοχή σε σελίδα JSP.

Λίστα 4.1 ThreeParams.java

```
package coreservlets;

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

/** Απλή μικροϋπηρεσία που διαβάζει τρεις παραμέτρους από τα
 *  δεδομένα φόρμας.
 */

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
```

Λίστα 4.1 ThreeParams.java *(συνέχεια)*

```
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Η Λίστα 4.2 δείχνει μια φόρμα HTML που συγκεντρώνει την είσοδο του χρήστη και τη στέλνει στη μικροϋπηρεσία. Αν χρησιμοποιήσετε στην ιδιότητα ACTION ένα URL που ξεκινά με κάθετο (/servlet/coreservlets.ThreeParams), θα μπορείτε να εγκαταστήσετε τη φόρμα HTML οπουδήποτε στην εφαρμογή Ιστού· μπορείτε να μεταφέρετε τη φόρμα HTML σε άλλον κατάλογο, ή να μεταφέρετε και τη φόρμα HTML και τη μικροϋπηρεσία σε άλλο μηχάνημα, χωρίς να χρειαστεί να αλλάξετε είτε τη φόρμα HTML είτε τη μικροϋπηρεσία. Η γενική αρχή ότι τα URL φορμών που ξεκινούν με κάθετο αυξάνουν τη φορητότητα ισχύει ακόμα και όταν χρησιμοποιείτε προσαρμοσμένες εφαρμογές Ιστού, αλλά θα πρέπει να συμπεριλάβετε στο URL το πρόθεμα της εφαρμογής Ιστού. Για λεπτομέρειες σχετικά με τις εφαρμογές Ιστού δείτε την Ενότητα 2.11 (Εφαρμογές Ιστού: Μια προεπισκόπηση). Υπάρχουν και άλλοι τρόποι για να γράψετε URL που διευκολύνουν τη φορητότητα, αλλά το πιο σημαντικό σημείο είναι να χρησιμοποιείτε σχετικά URL (χωρίς το όνομα του υπολογιστή υπηρεσίας) και όχι απόλυτα (όπως http://*υπολογιστής-Υπηρεσίας*/...) Αν χρησιμοποιήσετε μια απόλυτη διεύθυνση URL, θα πρέπει να διορθώνετε τις φόρμες όποτε μετακινείτε την εφαρμογή Ιστού από το ένα μηχάνημα στο άλλο. Επειδή σχεδόν σίγουρα η ανάπτυξη γίνεται σε ένα μηχάνημα και η εκδίπλωση σε κάποιο άλλο, η χρήση απόλυτων URL θα πρέπει να αποφεύγεται αυστηρά.

Η προσέγγιση του βιβλίου

Στην ιδιότητα ACTION της φόρμας να χρησιμοποιείτε σχετικά URL, και όχι απόλυτα.

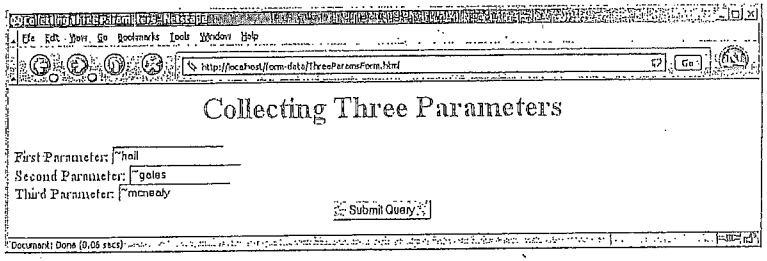
Λίστα 4.2 ThreeParamsForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Collecting Three Parameters</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<FORM ACTION="/servlet/coreservlets.ThreeParams">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>

</BODY></HTML>
```


Θυμηθείτε ότι η θέση της προεπιλεγμένης εφαρμογής Ιστού διαφέρει από διακομιστή σε διακομιστή. Οι φόρμες HTML πηγαίνουν στον κατάλογο ανώτατου επιπέδου ή σε υποκαταλόγους διαφορετικούς από τον WEB-INF. Αν τοποθετήσουμε τη σελίδα HTML στον υποκατάλογο form-data και την προσπελάσουμε από το τοπικό μηχάνημα, τότε η πλήρης θέση εγκατάστασης για τους τρεις διακομιστές που χρησιμοποιούνται στο βιβλίο είναι η εξής:

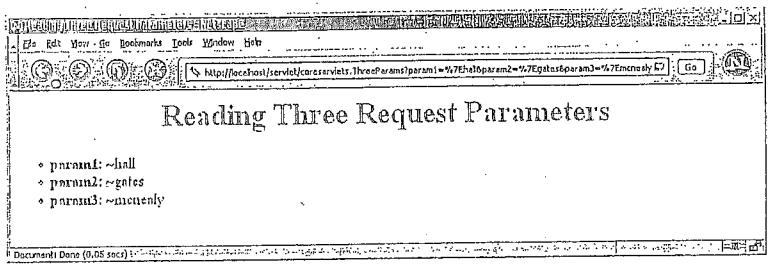
- **Θέση στον Tomcat**
  κατ_εγκατάστασης/webapps/ROOT/form-data/ThreeParams.html
- **Θέση στον JRun**
  κατ_εγκατάστασης/servers/default/default-ear/default-war/form-data/ThreeParams.html
- **Θέση στον Resin**
  κατ_εγκατάστασης/doc/form-data/ThreeParams.html
- **Αντίστοιχο URL**
  http://localhost/form-data/ThreeParams.html

Η Εικόνα 4-1 δείχνει τη φόρμα HTML όταν ο χρήστης έχει εισαγάγει τα ονόματα προσωπικών φακέλων για τρεις διάσημες προσωπικότητες του Internet. Εντάξει, εντάξει, μόνο δύο από αυτούς είναι διάσημοι,[1] αλλά το θέμα μας εδώ είναι ότι η περισπωμένη (~) είναι ένας μη αλφαριθμητικός χαρακτήρας και θα κωδικοποιηθεί από το φυλλομετρητή στο URL όταν υποβληθεί η φόρμα. Η Εικόνα 4-2 δείχνει το αποτέλεσμα της μικροϋπηρεσίας· προσέξτε τις κωδικοποιημένες τιμές στο URL που φαίνεται στη γραμμή διευθύνσεων αλλά και τη σωστή μορφή των τιμών των πεδίων στην έξοδο: η getParameter επιστρέφει πάντοτε τις τιμές όπως ακριβώς τις έχει πληκτρολογήσει ο χρήστης, ανεξάρτητα από τον τρόπο με τον οποίο στάλθηκαν οι τιμές αυτές στο δίκτυο.



Εικόνα 4-1 Οθόνη εισαγωγής δεδομένων για τη μικροϋπηρεσία επεξεργασίας παραμέτρων.

[1] Αν και ο Gates δεν είναι και *τόσο* διάσημος.



Εικόνα 4-2 Το αποτέλεσμα από τη μικροϋπηρεσία επεξεργασίας παραμέτρων: οι παράμετροι αίτησης έχουν αποκωδικοποιηθεί αυτόματα από το URL.

4.4 Παράδειγμα: Ανάγνωση όλων των παραμέτρων

Στο προηγούμενο παράδειγμα κάναμε εξαγωγή των τιμών των παραμέτρων από τα δεδομένα της φόρμας με βάση κάποια προκαθορισμένα ονόματα παραμέτρων. Θεωρήσαμε επίσης ότι η κάθε παράμετρος έχει μία μόνο τιμή. Θα δούμε τώρα ένα παράδειγμα που εξετάζει *όλα* τα ονόματα των παραμέτρων που έχουν σταλεί και τοποθετεί τις τιμές τους σε έναν πίνακα. Η εφαρμογή αυτή επισημαίνει τις παραμέτρους των οποίων οι τιμές λείπουν, καθώς επίσης και εκείνες που έχουν πολλές τιμές. Αν και αυτή η προσέγγιση χρησιμοποιείται σπάνια σε μικροϋπηρεσίες παραγωγής (αν δεν γνωρίζετε τα ονόματα των παραμέτρων μιας φόρμας, τότε μάλλον δεν θα γνωρίζετε τι να κάνετε με αυτές), είναι αρκετά χρήσιμη για αποσφαλμάτωση.

Καταρχήν η μικροϋπηρεσία παίρνει όλα τα ονόματα παραμέτρων με τη μέθοδο `getParameterNames` της κλάσης `HttpServletRequest`. Αυτή η μέθοδος επιστρέφει μια απαρίθμηση (`Enumeration`) που περιέχει τα ονόματα των παραμέτρων με τυχαία σειρά. Στη συνέχεια η μικροϋπηρεσία διατρέχει την απαρίθμηση με τον τυπικό τρόπο, χρησιμοποιώντας τη μέθοδο `hasMoreElements` για να καθορίσει πότε θα σταματήσει και τη μέθοδο `nextElement` για να πάρει το κάθε όνομα παραμέτρου. Επειδή η μέθοδος `nextElement` επιστρέφει ένα αντικείμενο (`Object`), η μικροϋπηρεσία μετατρέπει (cast) το αποτέλεσμα σε τύπο δεδομένων `String` και το μεταβιβάζει στη μέθοδο `getParameterValues`, παρέχοντας τελικά έναν πίνακα αλφαριθμητικών. Αν αυτός ο πίνακας έχει μέγεθος μία καταχώριση και περιέχει ένα κενό αλφαριθμητικό, τότε η παράμετρος δεν έχει τιμές και η μικροϋπηρεσία παράγει την καταχώριση "No Value" (Χωρίς τιμή) σε πλάγια γραφή. Αν ο πίνακας έχει μέγεθος μεγαλύτερο από μία καταχώριση, τότε η παράμετρος έχει πολλές τιμές οπότε αυτές εμφανίζονται σε μία λίστα με κουκκίδες. Διαφορετικά, η μοναδική τιμή τοποθετείται κατευθείαν στον πίνακα.

Ο πηγαίος κώδικας της μικροϋπηρεσίας φαίνεται στη Λίστα 4.3· η Λίστα 4.4 παρουσιάζει τον κώδικα HTML για μια σελίδα εισαγωγής δεδομένων που μπορείτε να χρησιμοποιήσετε για να δοκιμάσετε τη μικροϋπηρεσία. Οι Εικόνες 4-3 και 4-4 δείχνουν το αποτέλεσμα της σελίδας HTML και της μικροϋπηρεσίας, αντίστοιχα.

Λίστα 4.3 ShowParameters.java

```
package coreservlets;

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

/** Εμφανίζει όλες τις παραμέτρους που στάλθηκαν στη μικροϋπηρεσία
 *  είτε μέσω GET είτε μέσω POST. Επισημαίνει ιδιαίτερα τις παραμέτρους
 *  που δεν έχουν τιμή ή έχουν πολλές τιμές.
 */

public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    String title = "Reading All Request Parameters";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>Parameter Name<TH>Parameter Value(s)");
    Enumeration paramNames = request.getParameterNames();
    while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.print("<TR><TD>" + paramName + "\n<TD>");
      String[] paramValues =
        request.getParameterValues(paramName);
      if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
          out.println("<I>No Value</I>");
        else
          out.println(paramValue);
      } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) {
          out.println("<LI>" + paramValues[i]);
        }
        out.println("</UL>");
      }
    }
  }
```

Λίστα 4.6 ShowParameters.java *(συνέχεια)*

```
    out.println("</TABLE>\n</BODY></HTML>");
  }

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

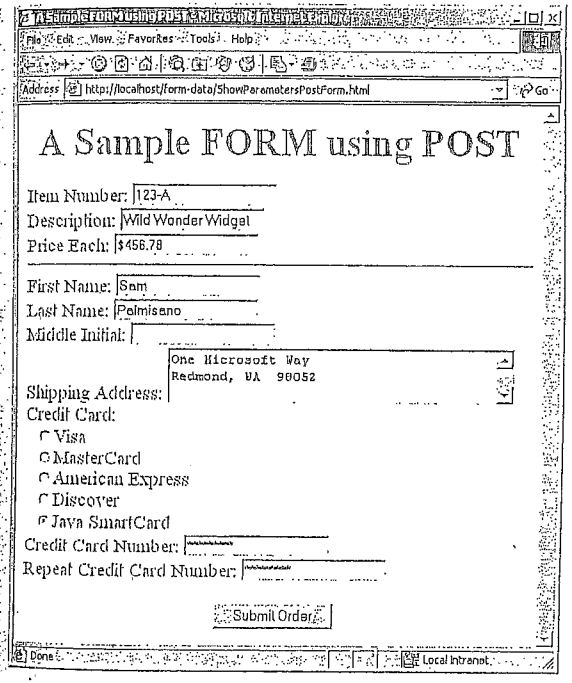
Παρατηρήστε ότι η μικροϋπηρεσία χρησιμοποιεί μια μέθοδο doPost η οποία απλώς καλεί τη μέθοδο doGet. Αυτό συμβαίνει επειδή θέλουμε να είμαστε σε θέση να χειριζόμαστε τόσο αιτήσεις GET όσο και αιτήσεις POST. Η προσέγγιση αυτή αποτελεί μια καλή τυπική πρακτική όταν θέλετε οι διασυνδέσεις HTML να διαθέτουν κάποια ευελιξία όσον αφορά τον τρόπο αποστολής δεδομένων στη μικροϋπηρεσία. Δείτε την περιγραφή της μεθόδου service στην Ενότητα 3.6 (Ο κύκλος ζωής μιας μικροϋπηρεσίας), όπου εξηγούμε γιατί είναι προτιμότερο η doPost να καλεί την doGet (ή το αντίστροφο) σε σύγκριση με τον άμεσο υποσκελισμό της μεθόδου service. Η φόρμα HTML της Λίστας 4.4 χρησιμοποιεί τη μέθοδο POST, όπως θα πρέπει να κάνουν όλες οι φόρμες που έχουν πεδία με κωδικούς πρόσβασης (για λεπτομέρειες δείτε το Κεφάλαιο 19, "Δημιουργία και επεξεργασία φορμών HTML"). Επειδή όμως η μικροϋπηρεσία ShowParameters δεν αναφέρεται ειδικά στη συγκεκριμένη σελίδα εισαγωγής δεδομένων, στην τοποθεσία Ιστού με την αρχειοθήκη πηγαίου κώδικα http://www.coreservlets.com/ μπορείτε να βρείτε μια παρόμοια φόρμα HTML η οποία χρησιμοποιεί την GET, έτσι ώστε να πειραματιστείτε μαζί της.

Λίστα 4.7 ShowParametersPostForm.html

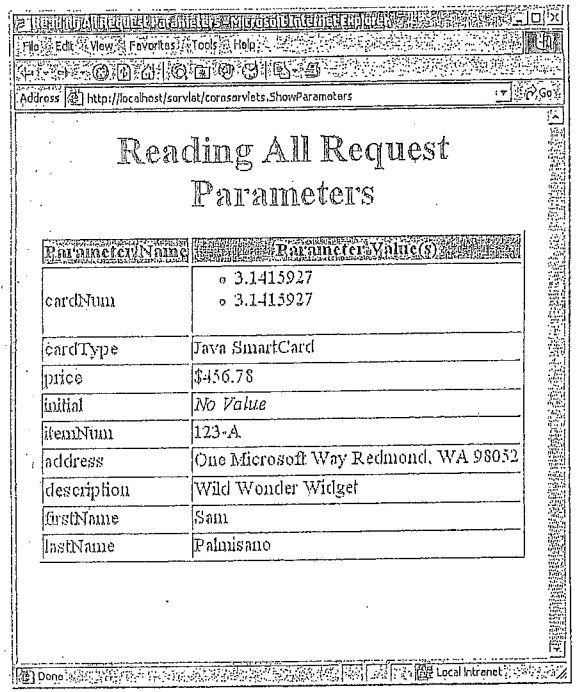
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>A Sample FORM using POST</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="/servlet/coreservlets.ShowParameters"
      METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Description: <INPUT TYPE="TEXT" NAME="description"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  <HR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  Last Name: <INPUT TYPE="TEXT" NAME="lastName"><BR>
  Middle Initial: <INPUT TYPE="TEXT" NAME="initial"><BR>
  Shipping Address:
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card:<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Visa">Visa<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
```


Λίστα 4.7 ShowParametersPostForm.html *(συνέχεια)*

```
                     VALUE="MasterCard">MasterCard<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Amex">American Express<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Discover">Discover<BR>
    <INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Java SmartCard">Java SmartCard<BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  Repeat Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR><BR>
  <CENTER><INPUT TYPE="SUBMIT" VALUE="Submit Order"></CENTER>
</FORM>
</BODY></HTML>
```



Εικόνα 4-3 Φόρμα HTML που συλλέγει δεδομένα για τη μικροϋπηρεσία ShowParameters.



Εικόνα 4-4 Το αποτέλεσμα της μικροϋπηρεσίας `ShowParameters`.

4.5 Χρήση προεπιλεγμένων τιμών για ελλιπείς ή εσφαλμένες παραμέτρους

Οι ηλεκτρονικές υπηρεσίες για ανεύρεση εργασίας έχουν αρχίσει τελευταία να αποκτούν μεγάλη δημοτικότητα. Μια αξιόπιστη τοποθεσία ιστού παρέχει χρήσιμες υπηρεσίες σε εκείνους που αναζητούν εργασία, αφού προβάλλει σε μεγάλο κοινό τις ικανότητές τους, αλλά παρέχει χρήσιμες υπηρεσίες και στους εργοδότες, δίνοντάς τους πρόσβαση σε μια μεγάλη "δεξαμενή" μελλοντικών υπαλλήλων. Αυτή η ενότητα παρουσιάζει μια μικροϋπηρεσία που χειρίζεται ένα τμήμα μιας τέτοιας τοποθεσίας Ιστού: τη δημιουργία ηλεκτρονικών βιογραφικών από δεδομένα που υποβάλλει ο χρήστης. Το ερώτημα που τίθεται τώρα είναι το εξής: τι θα πρέπει να κάνει η μικροϋπηρεσία αν ο χρήστης δεν δώσει τις απαραίτητες πληροφορίες; Αυτή η ερώτηση έχει δύο απαντήσεις: χρήση προεπιλεγμένων τιμών ή επανεμφάνιση της φόρμας (ζητώντας από το χρήστη να συμπληρώσει τις τιμές που λείπουν). Η ενότητα αυτή παρουσιάζει τη χρήση προεπιλεγμένων τιμών· η Ενότητα 4.8 παρουσιάζει τη λύση της επανεμφάνισης της φόρμας.

DILBERT Ανατύπωση με την άδεια της United Feature Syndicate, Inc.

Όταν εξετάζετε παραμέτρους αιτήσεων, θα πρέπει να ελέγχετε τρεις περιπτώσεις:

1. **Η τιμή είναι `null`.** Η κλήση της μεθόδου `request.getParameter` επιστρέφει τιμή `null` αν η φόρμα δεν περιέχει πεδίο κειμένου ή άλλο στοιχείο με το αναμενόμενο όνομα, οπότε το όνομα της παραμέτρου δεν εμφανίζεται καθόλου στην αίτηση. Αυτό μπορεί να συμβεί αν ο χρήστης χρησιμοποιήσει λανθασμένη φόρμα HTML ή αν χρησιμοποιηθεί ένα URL που έχει αποθηκευτεί ως σελιδοδείκτης και το όνομα της παραμέτρου έχει αλλάξει από τότε που αποθηκεύτηκε το URL. Για να αποφύγετε το σφάλμα `NullPointerException` (Εξαίρεση μηδενικού δείκτη), θα πρέπει να ελέγχετε για την παρουσία της τιμής `null` *πριν* προσπαθήσετε να καλέσετε οποιαδήποτε μέθοδο για το αλφαριθμητικό που προκύπτει από τη μέθοδο `getParameter`.
2. **Η τιμή είναι ένα κενό αλφαριθμητικό.** Η κλήση της μεθόδου `request.getParameter` επιστρέφει ένα κενό αλφαριθμητικό (δηλαδή `""`) αν το αντίστοιχο πεδίο κειμένου είναι κενό κατά την υποβολή της φόρμας. Για να ελέγξετε την παρουσία κενού αλφαριθμητικού, συγκρίνετε το αλφαριθμητικό με το `""` χρησιμοποιώντας τη μέθοδο `equals` ή συγκρίνετε το μήκος του αλφαριθμητικού με το 0. *Μη χρησιμοποιείτε τον τελεστή* `==`: στη γλώσσα προγραμματισμού Java, ο τελεστής `==` ελέγχει αν δύο ορίσματα αφορούν το ίδιο αντικείμενο (στην ίδια θέση στη μνήμη), και όχι αν είναι παρόμοια. Για να είστε ασφαλείς, καλή ιδέα είναι να χρησιμοποιήσετε την `trim` για να αφαιρέσετε τα τυχόν κενά διαστήματα που μπορεί να έχει πληκτρολογήσει ο χρήστης, αφού στα περισσότερα σενάρια θέλετε τα κενά διαστήματα να ισοδυναμούν με ελλιπή δεδομένα. Έτσι, για παράδειγμα, ο έλεγχος ελλιπών δεδομένων μπορεί να μοιάζει με το ακόλουθο τμήμα κώδικα.

```
String param = request.getParameter("someName");
if ((param == null) || (param.trim().equals(""))) {
  doSomethingForMissingValues(...);
} else {
  doSomethingWithParameter(param);
}
```

3. **Η τιμή είναι ένα μη κενό αλφαριθμητικό λανθασμένης μορφής.** Το τι αποτελεί λανθασμένη μορφή εξαρτάται από τη συγκεκριμένη εφαρμογή: μπορεί να περιμένετε τα πεδία κειμένου να περιέχουν μόνο αριθμητικές τιμές, σε άλλη περίπτωση να έχουν ακριβώς επτά χαρακτήρες, ή σε άλλη περίπτωση να περιέχουν μόνο γράμματα.

Πρέπει να σημειώσουμε ότι η χρήση της JavaScript για επικύρωση στην πλευρά του πελάτη δεν αναιρεί την ανάγκη πραγματοποίησης αυτού του ελέγχου στο διακομιστή. Έτσι κι αλλιώς, εσείς είστε υπεύθυνοι για την εφαρμογή στην πλευρά του διακομιστή και κάποιος άλλος προγραμματιστής ή κάποια άλλη ομάδα είναι υπεύθυνοι για τις φόρμες. Δεν θέλετε η *δική σας* εφαρμογή να καταρρεύσει αν *αυτοί* δεν μπορέσουν να ανιχνεύσουν κάθε τύπο μη έγκυρης εισόδου. Εξάλλου, οι πελάτες μπορούν είτε να χρησιμοποιήσουν δικές τους φόρμες HTML, είτε να αλλάξουν τις διευθύνσεις URL που περιέχουν δεδομένα GET, ή να απενεργοποιήσουν την JavaScript.

Η προσέγγιση του βιβλίου

Σχεδιάστε τις μικροϋπηρεσίες σας με τέτοιον τρόπο ώστε να χειρίζονται προσεκτικά τις παραμέτρους που λείπουν (null ή κενό αλφαριθμητικό) ή που έχουν μορφοποιηθεί λανθασμένα. Ελέγξτε τις μικροϋπηρεσίες σας τόσο με ελλιπή δεδομένα ή δεδομένα λανθασμένης μορφής, όσο και με δεδομένα που παρέχονται στην αναμενόμενη μορφή.

Η Λίστα 4.5 και η Εικόνα 4-5 παρουσιάζουν τη λειτουργία της φόρμας HTML ως "εμπροσθοφυλακής" για τη μικροϋπηρεσία που επεξεργάζεται τα βιογραφικά. Αν δεν είστε εξοικειωμένοι με τις φόρμες HTML, διαβάστε το Κεφάλαιο 19. Η φόρμα χρησιμοποιεί τη μέθοδο POST για να υποβάλει τα δεδομένα, και συγκεντρώνει τιμές για διάφορα ονόματα παραμέτρων. Το σημαντικό στοιχείο που πρέπει να καταλάβετε εδώ είναι το τι κάνει η μικροϋπηρεσία όταν λείπουν δεδομένα ή όταν αυτά έχουν λανθασμένη μορφή. Η διαδικασία αυτή συνοψίζεται στον κατάλογο που ακολουθεί. Η Λίστα 4.6 παρουσιάζει τον πλήρη κώδικα της μικροϋπηρεσίας.

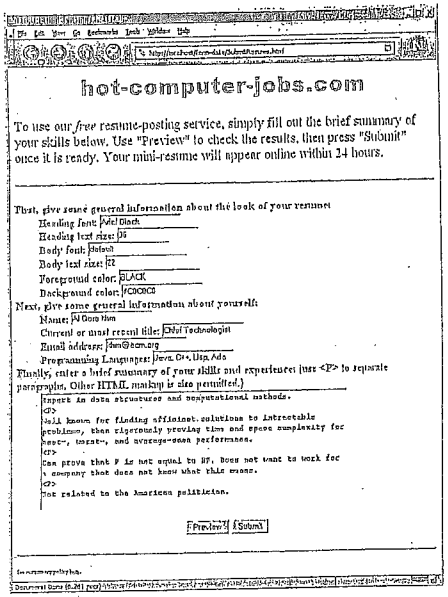
- `name, title, email, languages, skills`
  Αυτές οι παράμετροι προσδιορίζουν διάφορα μέρη του βιογραφικού. Αν απουσιάζουν τιμές, αυτές θα πρέπει να αντικαθίστανται με κατάλληλες προεπιλεγμένες τιμές ειδικά για κάθε παράμετρο. Για να επιτύχει αυτόν το στόχο, η μικροϋπηρεσία χρησιμοποιεί μια μέθοδο που ονομάζεται `replaceIfMissing`.
- `fgColor, bgColor`
  Αυτές οι παράμετροι καθορίζουν τα χρώματα προσκηνίου και παρασκηνίου της σελίδας. Αν απουσιάζουν τιμές αυτό ισοδυναμεί με μαύρο χρώμα για το προσκήνιο και λευκό για το φόντο. Η μικροϋπηρεσία χρησιμοποιεί και εδώ τη μέθοδο `replaceIfMissing` για να εκτελέσει αυτή την εργασία.
- `headingFont, bodyFont`
  Αυτές οι παράμετροι καθορίζουν τη γραμματοσειρά που θα χρησιμοποιηθεί για τις κεφαλίδες και το κυρίως κείμενο, αντίστοιχα. Η απουσία τιμής ή η τιμή "default" θα έχει αποτέλεσμα τη χρήση μιας γραμματοσειράς sans-serif, σαν την Arial ή τη Helvetica. Η μικροϋπηρεσία χρησιμοποιεί τη μέθοδο `replaceIfMissingOrDefault` για να επιτύχει αυτόν το σκοπό.
- `headingSize, bodySize`
  Αυτές οι παράμετροι καθορίζουν το μέγεθος της γραμματοσειράς για τις κύριες επικεφαλίδες και το κείμενο του σώματος της σελίδας, αντίστοιχα. Οι δευτερεύουσες επικεφαλίδες θα εμφανίζονται με ελαφρώς μικρότερο μέγεθος από τις κύριες επικεφαλίδες. Αν απουσιάζουν τιμές ή αν έχουν πληκτρολογηθεί μη αριθμητικές τιμές, τότε θα πρέπει να χρησιμοποιείται το προεπιλεγμένο μέγεθος (32 για τις επικεφαλίδες και 18 για το σώμα του κειμένου). Για το χειρισμό αυτής της περίπτωσης, η μικροϋπηρεσία χρησιμοποιεί μια κλήση της `Integer.parseInt` και ένα μπλοκ κώδικα `try/catch` για την εξαίρεση `NumberFormatException`.

Λίστα 4.5 SubmitResume.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Free Resume Posting</TITLE>
  <LINK REL=STYLESHEET
        HREF="jobs-site-styles.css"
        TYPE="text/css">
</HEAD>
<BODY>
<H1>hot-computer-jobs.com</H1>
<P CLASS="LARGER">
To use our <I>free</I> resume-posting service, simply fill
out the brief summary of your skills below. Use "Preview"
to check the results, then press "Submit" once it is
ready. Your mini-resume will appear online within 24 hours.</P>
<HR>
<FORM ACTION="/servlet/coreservlets.SubmitResume"
      METHOD="POST">
<DL>
<DT><B>First, give some general information about the look of
your resume:</B>
<DD>Heading font:
    <INPUT TYPE="TEXT" NAME="headingFont" VALUE="default">
<DD>Heading text size:
    <INPUT TYPE="TEXT" NAME="headingSize" VALUE=32>
<DD>Body font:
    <INPUT TYPE="TEXT" NAME="bodyFont" VALUE="default">
<DD>Body text size:
    <INPUT TYPE="TEXT" NAME="bodySize" VALUE=18>
<DD>Foreground color:
    <INPUT TYPE="TEXT" NAME="fgColor" VALUE="BLACK">
<DD>Background color:
    <INPUT TYPE="TEXT" NAME="bgColor" VALUE="WHITE">

<DT><B>Next, give some general information about yourself:</B>
<DD>Name: <INPUT TYPE="TEXT" NAME="name">
```

Λίστα 4.15 SubmitResume.html (συνέχεια)

```
<DD>Current or most recent title:
    <INPUT TYPE="TEXT" NAME="title">
<DD>Email address: <INPUT TYPE="TEXT" NAME="email">
<DD>Programming Languages:
    <INPUT TYPE="TEXT" NAME="languages">

<DT><B>Finally, enter a brief summary of your skills and
    experience:</B> (use &lt;P&gt; to separate paragraphs.
    Other HTML markup is also permitted.)
<DD><TEXTAREA NAME="skills"
              ROWS=10 COLS=60 WRAP="SOFT"></TEXTAREA>
</DL>
  <CENTER>
    <INPUT TYPE="SUBMIT" NAME="previewButton" Value="Preview">
    <INPUT TYPE="SUBMIT" NAME="submitButton" Value="Submit">
  </CENTER>
</FORM>
<HR>
<P CLASS="TINY">See our privacy policy
<A HREF="we-will-spam-you.html">here</A>.</P>
</BODY></HTML>
```



Εικόνα 4-5 Σελίδα "εμπροσθοφυλακής" για τη μικροϋπηρεσία επισκόπησης βιογραφικού.

Λίστα 4.16 SubmitResume.java

```
package coreservlets;

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

/** Μικροϋπηρεσία που χειρίζεται την προεπισκόπηση και αποθήκευση
 *  βιογραφικών που υποβάλλουν αυτοί που ζητούν εργασία.
 */

public class SubmitResume extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    if (request.getParameter("previewButton") != null) {
      showPreview(request, out);
    } else {
      storeResume(request);
      showConfirmation(request, out);
    }
  }

  /** Παρουσιάζει μια προεπισκόπηση του υποβληθέντος βιογραφικού.
   *  Παίρνει τις πληροφορίες γραμματοσειράς και φτιάχνει από αυτό
   *  ένα φύλλο στυλ HTML. Κατόπιν παίρνει τις πραγματικές πληροφορίες
   *  του βιογραφικού και τις παρουσιάζει μορφοποιημένες σύμφωνα με
   *  αυτό το φύλλο στυλ.
   */

  private void showPreview(HttpServletRequest request,
                           PrintWriter out) {
    String headingFont = request.getParameter("headingFont");
    headingFont = replaceIfMissingOrDefault(headingFont, "");
    int headingSize =
      getSize(request.getParameter("headingSize"), 32);
    String bodyFont = request.getParameter("bodyFont");
    bodyFont = replaceIfMissingOrDefault(bodyFont, "");
    int bodySize =
      getSize(request.getParameter("bodySize"), 18);
    String fgColor = request.getParameter("fgColor");
    fgColor = replaceIfMissing(fgColor, "BLACK");
    String bgColor = request.getParameter("bgColor");
    bgColor = replaceIfMissing(bgColor, "WHITE");
    String name = request.getParameter("name");
    name = replaceIfMissing(name, "Lou Zer");
    String title = request.getParameter("title");
    title = replaceIfMissing(title, "Loser");
```

Λίστα 4.6 SubmitResume.java *(συνέχεια)*

```
ing email = request.getParameter("email");
il =
eplaceIfMissing(email, "contact@hot-computer-jobs.com");
ing languages = request.getParameter("languages");
guages = replaceIfMissing(languages, "<I>None</I>");
ing languageList = makeList(languages);
ing skills = request.getParameter("skills");
lls = replaceIfMissing(skills, "Not many, obviously.");
.println
ervletUtilities.DOCTYPE + "\n" +
'<HTML><HEAD><TITLE>Resume for " + name + "</TITLE>\n" +
akeStyleSheet(headingFont, headingSize,
              bodyFont, bodySize,
              fgColor, bgColor) + "\n" +
'</HEAD>\n" +
'<BODY>\n" +
'<CENTER>\n"+
'<SPAN CLASS=\"HEADING1\">" + name + "</SPAN><BR>\n" +
'<SPAN CLASS=\"HEADING2\">" + title + "<BR>\n" +
'<A HREF=\"mailto:" + email + "\">" + email +
   "</A></SPAN>\n" +
</CENTER><BR><BR>\n" +
<SPAN CLASS=\"HEADING3\">Programming Languages" +
</SPAN>\n" +
akeList(languages) + "<BR><BR>\n" +
<SPAN CLASS=\"HEADING3\">Skills and Experience" +
</SPAN><BR><BR>\n" +
kills + "\n" +
</BODY></HTML>");


μεί ένα φύλλο επάλληλων στυλ (cascading style sheet) με
ηροφορίες σε τρία επίπεδα επικεφαλίδων και ολική κάλυψη
οσκηνίου και φόντου. Επίσης ζητά από τον Internet Explorer
αλλάζει το χρώμα του συνδέσμου mailto όταν το ποντίκι
ονά πάνω από αυτόν.


e String makeStyleSheet(String headingFont,
                        int heading1Size,
                        String bodyFont,
                        int bodySize,
                        String fgColor,
                        String bgColor) {
eading2Size = heading1Size*7/10;
eading3Size = heading1Size*6/10;
g styleSheet =
TYLE TYPE=\"text/css\">\n" +
--\n" +
EADING1 { font-size: " + heading1Size + "px;\n" +
```


Λίστα 4.6 SubmitResume.java *(συνέχεια)*

```
"             font-weight: bold;\n" +
"             font-family: " + headingFont +
                "Arial, Helvetica, sans-serif;\n" +
"}\n" +
".HEADING2 { font-size: " + heading2Size + "px;\n" +
"             font-weight: bold;\n" +
"             font-family: " + headingFont +
                "Arial, Helvetica, sans-serif;\n" +
"}\n" +
".HEADING3 { font-size: " + heading3Size + "px;\n" +
"             font-weight: bold;\n" +
"             font-family: " + headingFont +
                "Arial, Helvetica, sans-serif;\n" +
"}\n" +
"BODY { color: " + fgColor + ";\n" +
"       background-color: " + bgColor + ";\n" +
"       font-size: " + bodySize + "px;\n" +
"       font-family: " + bodyFont +
              "Times New Roman, Times, serif;\n" +
"}\n" +
"A:hover { color: red; }\n" +
"-->\n" +
"</STYLE>";
  return(styleSheet);
}

/** Αντικαθιστά αλφαριθμητικά null (δεν υπάρχει τέτοιο όνομα παραμέτρου)
 *  ή κενά αλφαριθμητικά (π.χ., το πεδίο κειμένου είναι κενό) με την
 *  τιμή αντικατάστασης. Διαφορετικά επιστρέφει το αρχικό αλφαριθμητικό
 */

private String replaceIfMissing(String orig,
                                String replacement) {
  if ((orig == null) || (orig.trim().equals(""))) {
    return(replacement);
  } else {
    return(orig);
  }
}

// Αντικαθιστά αλφαριθμητικά null, κενά αλφαριθμητικά, ή το
// αλφαριθμητικό "default" με την τιμή αντικατάστασης.
// Διαφορετικά επιστρέφει το αρχικό αλφαριθμητικό.

private String replaceIfMissingOrDefault(String orig,
                                         String replacement) {
  if ((orig == null) ||
      (orig.trim().equals("")) ||
```

Λίστα 4.6 SubmitResume.java *(συνέχεια)*

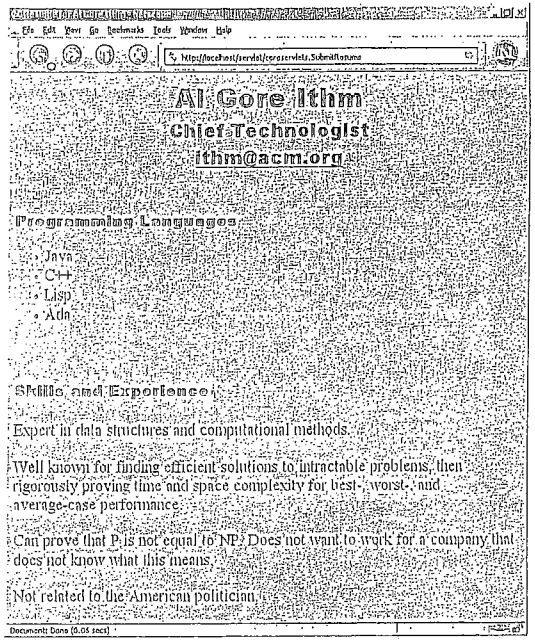
```
      (orig.equals("default"))) {
    return(replacement);
  } else {
    return(orig + ", ");
  }
}

// Παίρνει ένα αλφαριθμητικό που αντιπροσωπεύει έναν ακέραιο και το
// επιστρέφει σε τύπο int. Επιστρέφει την προεπιλεγμένη τιμή αν το
// αλφαριθμητικό είναι null ή δεν έχει έγκυρη μορφή.

private int getSize(String sizeString, int defaultSize) {
  try {
    return(Integer.parseInt(sizeString));
  } catch(NumberFormatException nfe) {
    return(defaultSize);
  }
}

// Αν δοθεί "Java,C++,Lisp", "Java C++ Lisp" ή
// "Java, C++, Lisp", επιστρέφει
// "<UL>
//   <LI>Java
//   <LI>C++
//   <LI>Lisp
// </UL>"

private String makeList(String listItems) {
  StringTokenizer tokenizer =
    new StringTokenizer(listItems, ", ");
  String list = "<UL>\n";
  while(tokenizer.hasMoreTokens()) {
    list = list + "  <LI>" + tokenizer.nextToken() + "\n";
  }
  list = list + "</UL>";
  return(list);
}

/** Παρουσιάζει μια σελίδα επιβεβαίωσης όταν ο χρήστης πατά
 *  με το ποντίκι στο κουμπί "Submit".
 */

private void showConfirmation(HttpServletRequest request,
                              PrintWriter out) {
  String title = "Submission Confirmed.";
  out.println(ServletUtilities.headWithTitle(title) +
              "<BODY>\n" +
              "<H1>" + title + "</H1>\n" +
              "Your resume should appear online within\n" +
              "24 hours. If it doesn't, try submitting\n" +
```


Λίστα 4.6 SubmitResume.java *(συνέχεια)*

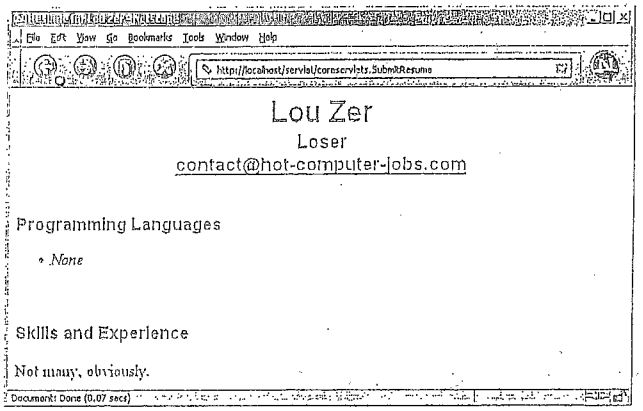
```
              "again with a different email address.\n" +
              "</BODY></HTML>");
  }

  /** Δεν είναι καλό να δίνετε τη διεύθυνση του ηλεκτρονικού σας
   *  ταχυδρομείου σε αναξιόπιστες τοποθεσίες Ιστού.
   */

  private void storeResume(HttpServletRequest request) {
    String email = request.getParameter("email");
    putInSpamList(email);
  }

  private void putInSpamList(String emailAddress) {
    // Ο κώδικας αφαιρέθηκε για να προστατευθεί ο ένοχος.
  }
}
```

Από τη στιγμή που η μικροϋπηρεσία έχει αποδεκτές τιμές για τις παραμέτρους γραμματοσειράς και χρώματος, δομεί από αυτές ένα φύλλο επάλληλων στυλ (cascading style sheet). Τα φύλλα στυλ είναι ο καθιερωμένος τρόπος καθορισμού των γραμματοσειρών, του μεγέθους των γραμματοσειρών, των χρωμάτων, των εσοχών, και άλλων πληροφοριών μορφοποίησης σε μια ιστοσελίδα HTML 4.0. Τα φύλλα στυλ συνήθως τοποθετούνται σε ένα ξεχωριστό αρχείο, έτσι ώστε διάφορες ιστοσελίδας μιας τοποθεσίας Ιστού να μπορούν να χρησιμοποιήσουν το ίδιο φύλλο στυλ, αλλά στη συγκεκριμένη περίπτωση είναι πιο βολική η ενσωμάτωση των πληροφοριών στυλ κατευθείαν στη σελίδα με χρήση του στοιχείου `STYLE`. Για περισσότερες πληροφορίες σχετικά με τα φύλλα στυλ, δείτε τη διεύθυνση http://www.w3.org/TR/REC-CSS1.

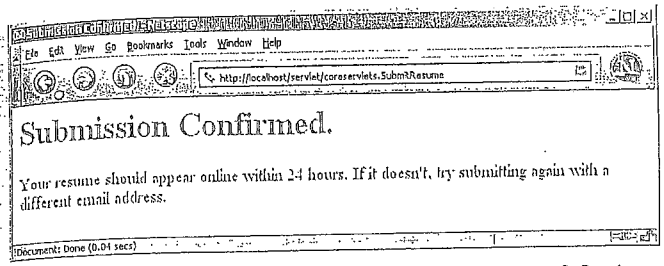
Μετά τη δημιουργία του φύλλου στυλ, η μικροϋπηρεσία τοποθετεί στην κορυφή της σελίδας το όνομα του αιτούντος, τον τίτλο της εργασίας, και τη διεύθυνση ηλεκτρονικού ταχυδρομείου, το ένα κάτω από το άλλο και στοιχισμένα στο κέντρο. Για αυτές τις γραμμές χρησιμοποιείται η γραμματοσειρά επικεφαλίδας και η διεύθυνση ηλεκτρονικού ταχυδρομείου τοποθετείται στο εσωτερικό του συνδέσμου υπερ-κειμένου `mailto:`, έτσι ώστε οι μελλοντικοί εργοδότες να μπορούν να επικοινωνήσουν με τον αιτούντα πατώντας με το ποντίκι σε αυτή τη διεύθυνση. Οι γλώσσες προγραμματισμού που καθορίζονται στην παράμετρο `languages` αναλύονται συντακτικά από τη μέθοδο `StringTokenizer` (με την υπόθεση ότι για το διαχωρισμό των ονομάτων των γλωσσών χρησιμοποιούνται διαστήματα ή κόμματα) και τοποθετούνται σε μια λίστα με κουκκίδες κάτω από την επικεφαλίδα "Programming Languages". Τέλος, το κείμενο της παραμέτρου `skills` τοποθετείται στο τέλος της σελίδας, κάτω από την επικεφαλίδα "Skills and Experience".



Εικόνα 4-6 Προεπισκόπηση της υποβολής βιογραφικού που περιλαμβάνει τα απαραίτητα δεδομένα φόρμας.



Εικόνα 4-7 Προεπισκόπηση υποβολής από την οποία λείπουν πολλά από τα απαραίτητα δεδομένα: οι τιμές που λείπουν αντικαταστάθηκαν από προεπιλεγμένες τιμές.



Εικόνα 4-8 Αποτέλεσμα υποβολής βιογραφικού στη βάση δεδομένων.

4.6 Φιλτράρισμα αλφαριθμητικών για ειδικούς χαρακτήρες HTML

Κανονικά, όταν μια μικροϋπηρεσία θέλει να παραγάγει κώδικα HTML ο οποίος θα περιέχει χαρακτήρες όπως οι < και >, χρησιμοποιεί απλώς τα αλφαριθμητικά `<` ή `>` που είναι οι αντίστοιχοι κωδικοί χαρακτήρων στην τυπική HTML. Παρομοίως, αν μια μικροϋπηρεσία θέλει να εμφανιστούν στην τιμή μιας ιδιότητας HTML διπλά εισαγωγικά ή ένα σύμβολο εμπορικού και (&), χρησιμοποιεί τους κωδικούς χαρακτήρων `&quot;` ή `&`. Αν δεν γίνουν αυτές οι αντικαταστάσεις, το αποτέλεσμα θα είναι κώδικας HTML με λανθασμένη μορφή, αφού οι χαρακτήρες < ή > συνήθως ερμηνεύονται ως ετικέτες σήμανσης HTML (markup tag), τα διπλά εισαγωγικά σε μια τιμή ιδιότητας μπορούν να θεωρηθούν ως το τέλος της τιμής, και τα σύμβολα εμπορικού και δεν είναι έγκυρες τιμές ιδιοτήτων. Στις περισσότερες περιπτώσεις είναι εύκολο να σημειώσουμε τους ειδικούς χαρακτήρες και να χρησιμοποιήσουμε στη θέση τους τυπικούς κωδικούς χαρακτήρων HTML. Υπάρχουν όμως δύο περιπτώσεις στις οποίες δεν είναι και τόσο εύκολο να κάνουμε αυτές τις αντικαταστάσεις.

Η πρώτη περίπτωση κατά την οποία η χειρονακτική μετατροπή είναι δύσκολη προκύπτει όταν το αλφαριθμητικό προέρχεται από κάποιο πρόγραμμα ή άλλη πηγή δεδομένων όπου είναι ήδη αποθηκευμένο με κάποια τυποποιημένη μορφή. Σε μια τέτοια περίπτωση η διαδικασία της εξέτασης και αλλαγής όλων των ειδικών χαρακτήρων μπορεί να είναι επίπονη, αλλά η παράλειψη ακόμα και ενός μόνο ειδικού χαρακτήρα μπορεί να κάνει την ιστοσελίδα σας να έχει ελλιπείς ή ακατάλληλα μορφοποιημένες ενότητες.

Η δεύτερη περίπτωση κατά την οποία η μη αυτόματη μετατροπή αποτυγχάνει είναι όταν το αλφαριθμητικό προκύπτει από δεδομένα φόρμας HTML. Εδώ η μετατροπή πρέπει υποχρεωτικά να γίνει κατά το χρόνο εκτέλεσης, αφού προφανώς τα δεδομένα ερωτήματος δεν είναι γνωστά κατά το χρόνο μεταγλώττισης. Αν ο χρήστης πληκτρολογήσει ακούσια ή εκούσια ετικέτες HTML, η ιστοσελίδα που θα παραχθεί θα έχει επίπλαστες ετικέτες HTML και μπορεί να δώσει τελείως απρόβλεπτα αποτελέσματα (οι προδιαγραφές της HTML λένε στους μεταγλωττιστές τι να κάνουν με τον έγκυρο κώδικα HTML· δεν λένε τίποτα για το τι θα πρέπει να κάνουν όταν η HTML περιέχει εσφαλμένη σύνταξη).

Η προσέγγιση του βιβλίου

Αν διαβάζετε παραμέτρους αίτησης και εμφανίζετε τις τιμές τους σε μια νέα σελίδα, θα πρέπει να φιλτράρετε τους ειδικούς χαρακτήρες HTML. Αν παραλείψετε να το κάνετε αυτό, τότε η έξοδος μπορεί να περιέχει ελλιπείς ή παράξενα μορφοποιημένες ενότητες.

Η παράλειψη αυτού του φιλτραρίσματος για ιστοσελίδες που προσπελάζονται εξωτερικά μπορεί να επιτρέψει στη σελίδα σας να αποτελέσει το μέσον για μια *επίθεση μεταξύ τοποθεσιών Ίσταν μέσω σεναρίου* (cross-site scripting attack). Εδώ, ένας κακόβουλος προγραμματιστής ενσωματώνει παραμέτρους GET σε μια διεύθυνση URL που αναφέρεται σε κάποια μικροϋπηρεσία σας (ή σε οποιοδήποτε άλλο πρόγραμμα που εκτελείται στο διακομιστή). Αυτές οι παράμετροι GET αναπτύσσονται σε ετικέτες HTML (συνήθως σε στοιχεία <SCRIPT>) που εκμεταλλεύονται γνωστά σφάλματα (bugs) των φυλλομετρητών. Έτσι ο επιτιθέμενος θα μπορούσε να ενσωματώσει κώδικα σε ένα URL που αναφέρεται στην τοποθεσία σας και να διανείμει μόνο το URL, και όχι την ίδια την κακόβουλη ιστοσελίδα. Με αυτόν τον τρόπο ο επιτιθέμενος θα μπορεί ευκολότερα να μην αποκαλυφθεί, και επίσης να αξιοποιήσει την εμπιστοσύνη των χρηστών για να τους κάνει να πιστέψουν ότι τα σενάρια προέρχονται από μια αξιόπιστη πηγή (την εταιρεία σας). Για περισσότερες λεπτομέρειες σχετικά με αυτό το θέμα, δείτε τις ιστοσελίδες http://www.cert.org/advisories/CA-2000-02.html και http://www.microsoft.com/technet/security/topics/ExSumCS.asp.

Κώδικας φιλτραρίσματος

Η αντικατάσταση των χαρακτήρων <, >, ", και & είναι απλή υπόθεση, που μπορεί να επιτευχθεί με διάφορες προσεγγίσεις. Ωστόσο, είναι σημαντικό να θυμάστε ότι τα αλφαριθμητικά της Java δεν μπορούν να τροποποιηθούν, οπότε η επαναλαμβανόμενη συνένωση αλφαριθμητικών περιλαμβάνει την αντιγραφή και μετά την απόρριψη πολλών τμημάτων αλφαριθμητικών. Για παράδειγμα, εξετάστε τις επόμενες δύο γραμμές κώδικα:

```
String s1 = "Hello";
String s2 = s1 + " World";
```

Επειδή το s1 δεν μπορεί να τροποποιηθεί, η δεύτερη γραμμή δημιουργεί ένα αντίγραφο του s1, προσαρτά τη λέξη "World" στο αντίγραφο, και μετά απορρίπτει το αντίγραφο. Για να αποφευχθεί η παραγωγή και η αντιγραφή αυτών των προσωρινών αντικειμένων, στις περιπτώσεις που θέλετε να πραγματοποιήσετε επαναλαμβανόμενες συνενώσεις στο εσωτερικό ενός βρόχου θα πρέπει να χρησιμοποιείτε κάποια δομή δεδομένων που να παρέχει τη δυνατότητα μεταβολής· η φυσική επιλογή είναι τα αντικείμενα StringBuffer.

Η προσέγγιση του βιβλίου

Αν θέλετε να κάνετε συνενώσεις αλφαριθμητικών στο εσωτερικό ενός βρόχου, χρησιμοποιήστε ένα αντικείμενο StringBuffer αντί για ένα αντικείμενο String.

Η Λίστα 4.7 παρουσιάζει τη στατική μέθοδο filter η οποία χρησιμοποιεί ένα αντικείμενο StringBuffer για να αντιγράψει αποδοτικά χαρακτήρες από το αλφαριθμητικό εισόδου σε μια φιλτραρισμένη εκδοχή του, αντικαθιστώντας στην πορεία τους τέσσερις ειδικούς χαρακτήρες.

Λίστα 4.7 ServletUtilities.java (Απόσπασμα)

```
package coreservlets;

import javax.servlet.*;
import javax.servlet.http.*;

public class ServletUtilities {
...

  /** Αντικατάσταση χαρακτήρων που έχουν ειδική σημασία στην HTML
   *  με τους αντίστοιχους κωδικούς χαρακτήρων HTML.
   */

  // Σημειώστε ότι δεν χρησιμοποιείται η Javadoc για πιο λεπτομερή
  // τεκμηρίωση, λόγω του ότι είναι δύσκολο να διαβαστούν οι ειδικοί
  // χαρακτήρες και σαν κείμενο και σαν HTML.
  //
  // Με δεδομένο ένα αλφαριθμητικό, αυτή η μέθοδος αντικαθιστά όλες
  // τις παρουσίες του '<' με '&lt;', όλες τις παρουσίες του '>' με
  // '&gt;', και (για το χειρισμό περιπτώσεων στο εσωτερικό τιμών
  // ιδιοτήτων) όλες τις παρουσίες διπλών εισαγωγικών με
  // '&quot;' και όλες τις παρουσίες του '&' με '&amp;'.
  // Χωρίς αυτό το φιλτράρισμα, δεν θα μπορεί να εισαχθεί με ασφάλεια
  // ένα τυχαίο αλφαριθμητικό σε μια ιστοσελίδα.

  public static String filter(String input) {
    if (!hasSpecialChars(input)) {
      return(input);
    }
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
      c = input.charAt(i);
      switch(c) {
        case '<': filtered.append("&lt;"); break;
        case '>': filtered.append("&gt;"); break;
        case '"': filtered.append("&quot;"); break;
        case '&': filtered.append("&amp;"); break;
        default: filtered.append(c);
      }
    }
    return(filtered.toString());
  }

  private static boolean hasSpecialChars(String input) {
    boolean flag = false;
```

Λίστα 4.7 ServletUtilities.java (Απόσπασμα) *(συνέχεια)*

```
    if ((input != null) && (input.length() > 0)) {
      char c;
      for(int i=0; i<input.length(); i++) {
        c = input.charAt(i);
        switch(c) {
          case '<': flag = true; break;
          case '>': flag = true; break;
          case '"': flag = true; break;
          case '&': flag = true; break;
        }
      }
    }
    return(flag);
  }
}
```

Παράδειγμα: Μικροϋπηρεσία που εμφανίζει αποσπάσματα κώδικα

Ως παράδειγμα, δείτε τη φόρμα HTML της Λίστας 4.8 που συγκεντρώνει ένα τμήμα κώδικα σε γλώσσα προγραμματισμού Java και το στέλνει στη μικροϋπηρεσία της Λίστα 4.9 για εμφάνιση.

Εδώ, όταν ο χρήστης πληκτρολογεί κανονική είσοδο, το αποτέλεσμα είναι μια χαρά, όπως φαίνεται στην Εικόνα 4-9. Ωστόσο, όπως φαίνεται στην Εικόνα 4-10, μπορεί να έχουμε απρόβλεπτα αποτελέσματα όταν ο χρήστης εισάγει ειδικούς χαρακτήρες, όπως οι < και >. Οι διάφοροι φυλλομετρητές μπορεί να δώσουν διάφορα αποτελέσματα, αφού οι προδιαγραφές της HTML δεν καθορίζουν τι πρέπει να γίνει σε αυτή την περίπτωση. Οι περισσότεροι φυλλομετρητές θα θεωρήσουν ότι το "<b" της παράστασης "if (a<b) {" σηματοδοτεί την αρχή μιας ετικέτας HTML. Επειδή όμως δεν αναγνωρίζονται οι χαρακτήρες μετά το b, οι φυλλομετρητές θα τους παραβλέψουν μέχρι τον επόμενο χαρακτήρα >, ο οποίος βρίσκεται στο τέλος της ετικέτας </PRE>. Έτσι όχι μόνο θα εξαφανιστεί το τμήμα κώδικα, αλλά ο φυλλομετρητής δεν θα ερμηνεύσει και την ετικέτα </PRE>, οπότε το κείμενο μετά το τμήμα κώδικα θα έχει λανθασμένη μορφοποίηση με γραμματοσειρά σταθερού πλάτους και απενεργοποιημένη την αναδίπλωση κειμένου.

Η Λίστα 4.10 παρουσιάζει μια μικροϋπηρεσία που λειτουργεί ακριβώς όπως και η προηγούμενη, με τη διαφορά ότι φιλτράρει τους ειδικούς χαρακτήρες από την τιμή της παραμέτρου αίτησης πριν να την εμφανίσει. Η Λίστα 4.11 δείχνει μια φόρμα HTML που στέλνει τα δεδομένα στη μικροϋπηρεσία (εκτός από το URL στην ιδιότητα ACTION, η φόρμα αυτή είναι ίδια με τη φόρμα της Λίστας 4.8). Η Εικόνα 4-11 δείχνει το αποτέλεσμα της εισόδου που προκάλεσε πρόβλημα με την προηγούμενη μικροϋπηρεσία: εδώ δεν υπάρχει κανένα πρόβλημα.

Λίστα 4.8 CodeForm1.html

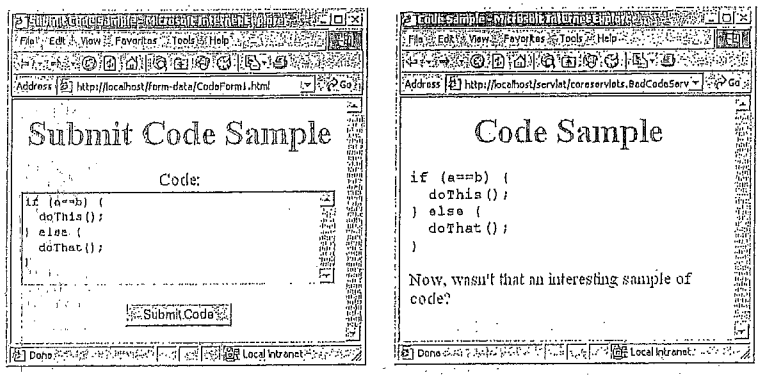
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Submit Code Sample</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1 ALIGN="CENTER">Submit Code Sample</H1>
<FORM ACTION="/servlet/coreservlets.BadCodeServlet">
 Code:<BR>
 <TEXTAREA ROWS="6" COLS="40" NAME="code"></TEXTAREA><P>
 <INPUT TYPE="SUBMIT" VALUE="Submit Code">
</FORM>
</CENTER></BODY></HTML>
```

Λίστα 4.9 BadCodeServlet.java

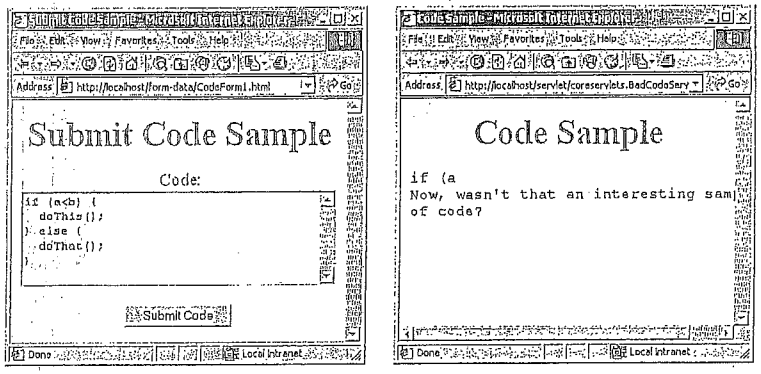
```
package coreservlets;

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

/** Μικροϋπηρεσία που διαβάζει ένα κομμάτι κώδικα από την αίτηση
 *  και το εμφανίζει μέσα σε μια ετικέτα PRE. Η μικροϋπηρεσία δεν
 *  εκτελεί φιλτράρισμα των ειδικών χαρακτήρων HTML.
 */

public class BadCodeServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Code Sample";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +

                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<PRE>\n" +
                getCode(request) +
                "</PRE>\n" +
                "Now, wasn't that an interesting sample\n" +
                "of code?\n" +
                "</BODY></HTML>");
  }
```

Λίστα 4.9 BadCodeServlet.java *(συνέχεια)*

```
  protected String getCode(HttpServletRequest request) {
    return(request.getParameter("code"));
  }
}
```



Εικόνα 4-9 BadCodeServlet: το αποτέλεσμα είναι σωστό όταν οι παράμετροι αίτησης δεν περιέχουν ειδικούς χαρακτήρες.



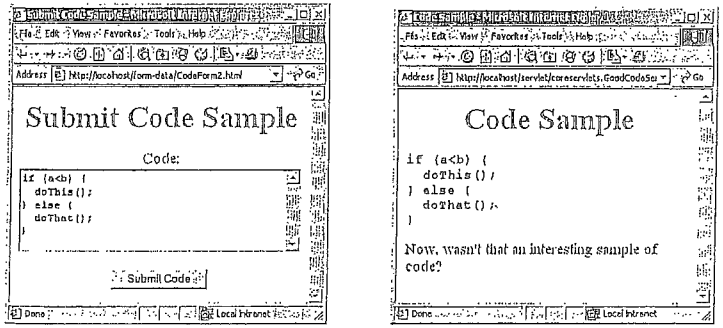
Εικόνα 4-10 BadCodeServlet: το αποτέλεσμα έχει ελλιπείς ενότητες και ενότητες με λανθασμένη μορφοποίηση όταν οι παράμετροι αίτησης περιέχουν ειδικούς χαρακτήρες.

Λίστα 4.10 GoodCodeServlet.java

```
package coreservlets;

import javax.servlet.http.*;

/** Μικρούπηρεσία που διαβάζει ένα κομμάτι κώδικα από την αίτηση
 *  και το εμφανίζει μέσα σε μία ετικέτα PRE. Η μικρούπηρεσία φιλτράρει
 *  τους ειδικούς χαρακτήρες HTML.
 */

public class GoodCodeServlet extends BadCodeServlet {
  protected String getCode(HttpServletRequest request) {
    return(ServletUtilities.filter(super.getCode(request)));
  }
}
```

Λίστα 4.11 CodeForm2.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Submit Code Sample</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1 ALIGN="CENTER">Submit Code Sample</H1>
<FORM ACTION="/servlet/coreservlets.GoodCodeServlet">
  Code:<BR>
  <TEXTAREA ROWS="6" COLS="40" NAME="code"></TEXTAREA><P>
  <INPUT TYPE="SUBMIT" VALUE="Submit Code">
</FORM>
</CENTER></BODY></HTML>
```



Εικόνα 4-11 GoodCodeServlet: το αποτέλεσμα είναι σωστό ακόμη και όταν οι παράμετροι της αίτησης περιέχουν ειδικούς χαρακτήρες.

4.7 Αυτόματη συμπλήρωση αντικειμένων Java: Κόκκοι φόρμας

Η μέθοδος `getParameter` κάνει εύκολη την ανάγνωση των εισερχόμενων παραμέτρων μιας αίτησης: χρησιμοποιείτε την ίδια μέθοδο τόσο στην `doGet` όσο και στην `doPost`, και η τιμή που επιστρέφεται αποκωδικοποιείται αυτόματα (δηλαδή, επιστρέφεται στη μορφή που πληκτρολογήθηκε από το χρήστη, και όχι υποχρεωτικά με τη μορφή που διακινήθηκε στο δίκτυο). Επειδή όμως η επιστρεφόμενη τιμή είναι τύπου `String`, αν θέλετε να πάρετε τιμές σε άλλους τύπους δεδομένων θα πρέπει να αναλύσετε μόνοι σας την τιμή (αφού φυσικά κάνετε έλεγχο για τιμές που λείπουν ή έχουν λανθασμένη μορφή). Για παράδειγμα, αν περιμένετε τιμές `int` ή `double`, θα πρέπει να μεταβιβάσετε το αποτέλεσμα της `getParameter` στη μέθοδο `Integer.parseInt` ή στη μέθοδο `Double.parseDouble` και να τοποθετήσετε τον κώδικα στο εσωτερικό ενός μπλοκ `try/catch` για την περίπτωση εξαίρεσης του τύπου `NumberFormatException`. Αν έχετε πολλές παραμέτρους αίτησης, αυτή η διαδικασία μπορεί να είναι αρκετά επίπονη.

Υποθέστε, για παράδειγμα, ότι έχετε ένα αντικείμενο δεδομένων με τρία πεδία `String`, δύο πεδία `int`, δύο πεδία `double`, και ένα πεδίο `boolean`. Η συμπλήρωση του αντικειμένου με βάση τα στοιχεία από μια υποβολή φόρμας απαιτεί οκτώ ξεχωριστές κλήσεις της `getParameter`, από δύο κλήσεις των `Integer.parseInt` και `Double.parseDouble`, και κάποιον ειδικό κώδικα για τη ρύθμιση της σημαίας `boolean`. Τι ωραία που θα ήταν να γίνεται αυτή η δουλειά αυτόματα!

Στις σελίδες JSP μπορείτε να χρησιμοποιήσετε την αρχιτεκτονική στοιχείων JavaBeans για να απλοποιήσετε σε μεγάλο βαθμό τη διαδικασία ανάγνωσης των παραμέτρων αίτησης, ανάλυσης των τιμών, και αποθήκευσης των αποτελεσμάτων σε αντικείμενα Java. Η διαδικασία αυτή περιγράφεται με λεπτομέρειες στο Κεφάλαιο 14 (Χρήση στοιχείων JavaBeans σε έγγραφα JSP). Αν δεν είστε εξοικειωμένοι με την ιδέα των κόκκων (beans), ανατρέξτε σε αυτό το κεφάλαιο. Η ουσία, όμως, είναι ότι ένα κανονικό αντικείμενο Java θεωρείται ότι είναι *κόκκος* αν η κλάση χρησιμοποιεί ιδιωτικά πεδία (private fields) και έχει μεθόδους που ακολουθούν τη σύμβαση ονομασίας get/set. Τα ονόματα των μεθόδων (χωρίς τις λέξεις "get" ή "set" και με το πρώτο γράμμα πεζό) ονομάζονται *ιδιότητες* (properties). Για παράδειγμα, μια κλάση Java που διαθέτει τις μεθόδους `getName` και `setName` λέμε ότι ορίζει έναν κόκκο ο οποίος έχει την ιδιότητα `name`.

Όπως περιγράφεται στο Κεφάλαιο 14, υπάρχει μια ειδική σύνταξη JSP (`property="*"` σε μια κλήση `jsp:setProperty`) την οποία μπορείτε να χρησιμοποιήσετε για να συμπληρώσετε με μία κίνηση έναν κόκκο με τιμές. Ειδικότερα, αυτή η ρύθμιση υποδεικνύει ότι το σύστημα θα πρέπει να εξετάσει όλες τις παραμέτρους της εισερχόμενης αίτησης και να τις μεταβιβάσει στις ιδιότητες του κόκκου που ταιριάζουν με το όνομα της παραμέτρου αίτησης. Πιο συγκεκριμένα, αν η παράμετρος αίτησης ονομάζεται `param1`, η παράμετρος μεταβιβάζεται στη μέθοδο `setParam1` του αντικειμένου. Επιπλέον, οι μετατροπές τύπων πραγματοποιούνται αυτόματα. Αν, για παράδειγμα, υπάρχει μια παράμετρος αίτησης που ονομάζεται `numOrdered` και το αντικείμενο διαθέτει μια μέθοδο που ονομάζεται `setNumOrdered` η οποία αναμένει τιμή `int` (δηλαδή ο κόκκος έχει την ιδιότητα `numOrdered` τύπου `int`), η παράμετρος αίτησης `numOrdered` μετατρέπεται αυτόματα σε τύπο δεδομένων `int` και η τιμή που προκύπτει μεταβιβάζεται αυτόματα στη μέθοδο `setNumOrdered`.

Αν μπορείτε να το κάνετε αυτό με τις σελίδες JSP, θα πιστεύετε ότι μπορείτε να το κάνετε και με τις μικροϋπηρεσίες. Έτσι κι αλλιώς, όπως εξηγούμε στο Κεφάλαιο 10, οι σελίδες JSP είναι στην πραγματικότητα "μεταμφιεσμένες" μικροϋπηρεσίες: κάθε σελίδα JSP μεταφράζεται σε μικροϋπηρεσία και κατά το χρόνο αίτησης εκτελείται αυτή η μικροϋπηρεσία. Επιπλέον, όπως θα δούμε στο Κεφάλαιο 15 "Ολοκλήρωση μικροϋπηρεσιών και σελίδων JSP: Η αρχιτεκτονική Model View Controller (MVC)", στα περίπλοκα σενάρια είναι συχνά καλύτερος ο συνδυασμός μικροϋπηρεσιών και σελίδων JSP με τέτοιον τρόπο ώστε οι μικροϋπηρεσίες να εκτελούν την προγραμματιστική εργασία ενώ οι σελίδες JSP να κάνουν τη δουλειά της παρουσίασης. Έτσι είναι πιο σημαντικό για τις μικροϋπηρεσίες να είναι σε θέση να διαβάζουν εύκολα τις παραμέτρους των αιτήσεων, απ' ό,τι είναι για τις σελίδες JSP. Το εντυπωσιακό, όμως, είναι ότι οι προδιαγραφές των μικροϋπηρεσιών δεν παρέχουν αυτή τη δυνατότητα: ο κώδικας που βρίσκεται πίσω από τη διεργασία `property="*"` των σελίδων JSP δεν είναι διαθέσιμος μέσω κάποιας καθιερωμένης API.

Ευτυχώς, το ευρέως χρησιμοποιούμενο πακέτο Jakarta Commons (δείτε τη διεύθυνση http://jakarta.apache.org/commons/) της "The Apache Software Foundation" περιέχει κλάσεις που διευκολύνουν τη δημιουργία μιας βοηθητικής κλάσης για την αυτόματη συσχέτιση παραμέτρων αιτήσεων με ιδιότητες κόκκων (δηλαδή με μεθόδους `setXxx`). Η επόμενη υποενότητα δίνει πληροφορίες για τη λήψη των πακέτων Commons, αλλά το σημαντικό σημείο είναι ότι μια στατική μέθοδος `populateBeans` μπορεί να δεχτεί ως είσοδο έναν κόκκο (δηλαδή, ένα αντικείμενο Java με τουλάχιστον μερικές μεθόδους που ακολουθούν τη σύμβαση ονομασίας get/set) και ένα αντικείμενο `Map` και να μεταβιβάσει όλες τις τιμές `Map` στην ιδιότητα του κόκκου που ταυτίζεται με το αντίστοιχο όνομα κλειδιού του αντικειμένου `Map`. Επιπρόσθετα, αυτή η βοηθητική κλάση επιτρέπει την αυτόματη μετατροπή τύπων με χρήση προεπιλεγμένων τιμών (π.χ. 0 για αριθμητικές τιμές), αντί να προκαλεί εξαίρεση όταν η αντίστοιχη παράμετρος αίτησης έχει λανθασμένη μορφή. Αν ο κόκκος δεν έχει ιδιότητα που να ταυτίζεται με το όνομα, η καταχώριση `Map` δεν λαμβάνεται υπόψη· και πάλι δεν προκαλείται κάποια εξαίρεση.

Η Λίστα 4.12 παρουσιάζει μια βοηθητική κλάση που χρησιμοποιεί το πακέτο Jakarta Commons για την αυτόματη συμπλήρωση ενός κόκκου σύμφωνα με τις παραμέτρους μιας εισερχόμενης αίτησης. Για να τη χρησιμοποιήσετε, μεταβιβάζετε απλώς τον κόκκο και το αντικείμενο της αίτησης στη μέθοδο `BeanUtilities.populateBean`. Αυτό είναι όλο. Θέλετε να τοποθετήσετε δύο παραμέτρους αίτησης σε ένα αντικείμενο δεδομένων; Κανένα πρόβλημα: το μόνο που χρειάζεται να κάνετε είναι μια κλήση της μεθόδου. Έχετε δεκαπέντε παραμέτρους αίτησης συν μερικές μετατροπές τύπου δεδομένων; Και πάλι κάνετε μόνο αυτή την κλήση της μεθόδου.

Λίστα 4.12 BeanUtilities.java

```
package coreservlets.beans;

import java.util.*;
import javax.servlet.http.*;
import org.apache.commons.beanutils.BeanUtils;

/** Μερικές βοηθητικές κλάσεις για τη συμπλήρωση κόκκων, συνήθως με
 *  βάση τις παραμέτρους μιας εισερχόμενης αίτησης. Απαιτεί τρία
```

Λίστα 4.12 BeanUtilities.java *(συνέχεια)*

```
 *  πακέτα της βιβλιοθήκης Apache Commons: τα beanutils, collections,
 *  και logging. Για να πάρετε αυτά τα πακέτα, επισκεφθείτε τη
 *  διεύθυνση http://jakarta.apache.org/commons/. Επίσης, η αρχειοθήκη
 *  πηγαίου κώδικα του βιβλίου (στη διεύθυνση http://www.coreservlets.com/)
 *  περιέχει συνδέσμους για όλα τα URL που αναφέρονται στο βιβλίο,
 *  καθώς και τις συγκεκριμένες ενότητες για το πακέτο Jakarta
 *  Commons.<P>
 *  Προσέξτε ότι αυτή η κλάση βρίσκεται στο πακέτο coreservlets.beans,
 *  οπότε πρέπει να εγκατασταθεί στον κατάλογο.../coreservlets/beans/.
public class BeanUtilities {
  /** Εξετάζει όλες τις παραμέτρους της αίτησης για να δει αν
   *  ταυτίζονται με κάποιες ιδιότητες του κόκκου (δηλαδή, με μια
   *  μέθοδο setXxx) του αντικειμένου. Αν συμβαίνει αυτό, τότε η τιμή
   *  της παραμέτρου αίτησης μεταβιβάζεται στη μέθοδο. Αν η μέθοδος
   *  αναμένει τιμή int, Integer, double, Double, ή οποιονδήποτε άλλο
   *  βασικό ή αναπτυγμένο τύπο δεδομένων, πραγματοποιείται αυτόματα
   *  ανάλυση και μετατροπή. Αν η τιμή της παραμέτρου αίτησης έχει
   *  λανθασμένη μορφή (δεν μπορεί να μετατραπεί στον αναμένόμενο τύπο
   *  δεδομένων), στις αριθμητικές ιδιότητες εκχωρείται τιμή μηδέν και
   *  στις ιδιότητες boolean εκχωρείται τιμή false: δεν προκαλείται
   *  εξαίρεση.
   */

  public static void populateBean(Object formBean,
                                  HttpServletRequest request) {
    populateBean(formBean, request.getParameterMap());
  }

  /** Συμπληρώνει έναν κόκκο με βάση ένα αντικείμενο Map: τα κλειδιά
   *  του αντικειμένου Map είναι τα ονόματα των ιδιοτήτων του κόκκου.
   *  οι τιμές του αντικειμένου Map είναι οι τιμές των ιδιοτήτων του
   *  κόκκου. Η μετατροπή τύπων δεδομένων γίνεται αυτόματα, όπως
   *  είδαμε παραπάνω.
   */

  public static void populateBean(Object bean,
                                  Map propertyMap) {
    try {
      BeanUtils.populate(bean, propertyMap);
    } catch(Exception e) {
      // Κενό σώμα της catch. Οι δύο πιθανές εξαιρέσεις είναι
      // java.lang.IllegalAccessException και
      // java.lang.reflect.InvocationTargetException. Και στις δύο
      // περιπτώσεις, παραλείπεται η αντίστοιχη λειτουργία του κόκκου.
    }
  }
}
```


Χρήση της κλάσης BeanUtilities

Η Λίστα 4.13 παρουσιάζει μια μικροϋπηρεσία που συγκεντρώνει ασφαλιστικές πληροφορίες για έναν υπάλληλο, πιθανόν για να χρησιμοποιηθούν έτσι ώστε να καθοριστούν τα διαθέσιμα ασφαλιστικά προγράμματα και το αντίστοιχο κόστος. Για την εκτέλεση αυτής της εργασίας, η μικροϋπηρεσία πρέπει να συμπληρώσει ένα αντικείμενο δεδομένων με πληροφορίες ασφάλισης (InsuranceInfo.java, Λίστα 4.14) σχετικά με το όνομα και τον αριθμό ταυτότητας του υπαλλήλου (και τα δύο είναι τύπου `String`), τον αριθμό των τέκνων (`int`), και το αν ο υπάλληλος είναι παντρεμένος ή όχι (`boolean`). Επειδή το αντικείμενο αναπαριστάνεται ως κόκκος, μπορεί να χρησιμοποιηθεί η μέθοδος `BeanUtilities.populateBean` για τη συμπλήρωση των απαιτούμενων πληροφοριών με κλήση μίας μόνο μεθόδου. Η Λίστα 4.15 δείχνει τη φόρμα HTML που συγκεντρώνει τα δεδομένα· οι Εικόνες 4-12 και 4-13 δείχνουν μερικά τυπικά αποτελέσματα.

Λίστα 4.13 SubmitInsuranceInfo.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import coreservlets.beans.*;

/** Παράδειγμα απλοποιημένης επεξεργασίας φόρμας. Δείχνει τη χρήση
 *  της BeanUtilities.populateBean για την αυτόματη συμπλήρωση ενός
 *  κόκκου (αντικειμένου Java με μεθόδους που ακολουθούν τις συμβάσεις
 *  ονομασίας get/set) από παραμέτρους αίτησης.
*/

public class SubmitInsuranceInfo extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    InsuranceInfo info = new InsuranceInfo();
    BeanUtilities.populateBean(info, request);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    String title = "Insurance Info for " + info.getName();
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<CENTER>\n" +
                "<H1>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI>Employee ID: " +
```

Λίστα 4.13 SubmitInsuranceInfo.java *(συνέχεια)*

```
                info.getEmployeeID() + "\n" +
              "  <LI>Number of children: " +
                info.getNumChildren() + "\n" +
              "  <LI>Married?: " +
                info.isMarried() + "\n" +
              "</UL></CENTER></BODY></HTML>");
  }
}
```

Λίστα 4.14 InsuranceInfo.java

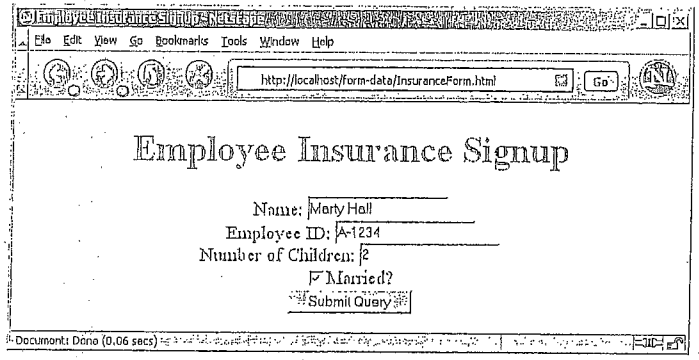
```
package coreservlets.beans;

import coreservlets.*;

/** Απλός κόκκος που αντιπροσωπεύει πληροφορίες απαραίτητες για
 *  τον υπολογισμό του κόστους ασφάλισης ενός υπαλλήλου. Έχει ιδιότητες
 *  τύπου String, int, και boolean. Χρησιμοποιείται για την επίδειξη
 *  της αυτόματης συμπλήρωσης ιδιοτήτων κόκκου από παραμέτρους αίτησης.
 */

public class InsuranceInfo {
  private String name = "No name specified";
  private String employeeID = "No ID specified";
  private int numChildren = 0;
  private boolean isMarried = false;

  public String getName() {
    return(name);
  }

  /** Στην περίπτωση που ο χρήστης πληκτρολογήσει ειδικούς χαρακτήρες
HTML,
   *  γίνεται φιλτράρισμά τους πριν από την αποθήκευση του ονόματος.
   */

  public void setName(String name) {
    this.name = ServletUtilities.filter(name);
  }

  public String getEmployeeID() {
    return(employeeID);
  }
```


Λίστα 4.14 InsuranceInfo.java *(συνέχεια)*

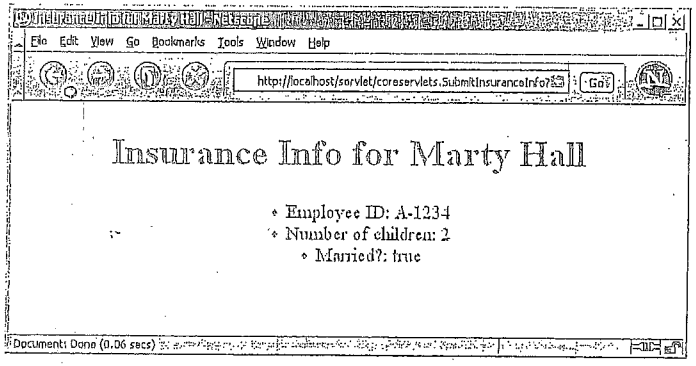
```
  /** Στην περίπτωση που ο χρήστης πληκτρολογήσει ειδικούς χαρακτήρες
   *  HTML, γίνεται φιλτράρισμά τους πριν από την αποθήκευση του
   *  ονόματος.
   */

  public void setEmployeeID(String employeeID) {
    this.employeeID = ServletUtilities.filter(employeeID);
  }

  public int getNumChildren() {
    return(numChildren);
  }

  public void setNumChildren(int numChildren) {
    this.numChildren = numChildren;
  }

  /** Σύμβαση κόκκου: το όνομα της μεθόδου είναι "isXxx" αντί
   *  για "getXxx" στις μεθόδους boolean.
   */

  public boolean isMarried() {
    return(isMarried);
  }

  public void setMarried(boolean isMarried) {
    this.isMarried = isMarried;
  }
}
```

Λίστα 4.15 InsuranceForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Employee Insurance Signup</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Employee Insurance Signup</H1>

<FORM ACTION="/servlet/coreservlets.SubmitInsuranceInfo">
  Name:  <INPUT TYPE="TEXT" NAME="name"><BR>
  Employee ID: <INPUT TYPE="TEXT" NAME="employeeID"><BR>
  Number of Children:  <INPUT TYPE="TEXT" NAME="numChildren"><BR>
  <INPUT TYPE="CHECKBOX" NAME="married" VALUE="true">Married?<BR>
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>

</CENTER></BODY></HTML>
```



Εικόνα 4-12 Εφαρμογή εισαγωγής δεδομένων για τη μικροϋπηρεσία επεξεργασίας ασφαλιστικών πληροφοριών.



Εικόνα 4-13 Μικροϋπηρεσία επεξεργασίας ασφαλιστικών πληροφοριών: η συγκέντρωση των δεδομένων αίτησης απλοποιείται σε μεγάλο βαθμό χάρη στη χρήση της μεθόδου `BeanUtilities.populateBean`.

Λήψη και εγκατάσταση των πακέτων Jakarta Commons

Η περισσότερη δουλειά στην κλάση `BeanUtilities` πραγματοποιείται από το στοιχείο `BeanUtils` του πακέτου Jakarta Commons. Αυτό το στοιχείο πραγματοποιεί την αντιστοίχιση — δηλαδή προσδιορίζει ποιες ιδιότητες κόκκου (μεθόδους `setXxx`) διαθέτει το αντικείμενο, στις οποίες μπορεί να γίνει καταχώριση — καθώς και τη μετατροπή τύπου δεδομένων (μετατροπή ενός `String` σε `int`, `double`, `boolean`, ή κάποιον άλλον θεμελιώδη ή παράγωγο τύπο). Έτσι, η κλάση `BeanUtilities` δεν θα λειτουργήσει αν δεν εγκαταστήσετε το πακέτο `BeanUtils` της Jakarta Commons. Επειδή όμως η `BeanUtils` βασίζεται σε δύο άλλα στοιχεία του Jakarta Commons— τα στοιχεία `Collections` και `Logging` — θα πρέπει να κατεβάσετε και να εγκαταστήσετε και τα τρία αυτά στοιχεία.

Για να κατεβάσετε αυτά τα στοιχεία, ξεκινήστε από τη διεύθυνση http://jakarta.apache.org/commons/, αναζητήστε την επικεφαλίδα "Components Repository" (Αποθήκη στοιχείων) στην αριστερή στήλη, και για καθένα από τα τρία αυτά στοιχεία κατεβάστε το αρχείο JAR που περιέχει την τελευταία έκδοση. (Ο κώδικάς μας βασίζεται στην έκδοση 1.5 της `BeanUtils`, αλλά το πιθανότερο είναι ότι και κάποια νεότερη έκδοση θα λειτουργεί παρόμοια.) Ο ευκολότερος ίσως τρόπος για να κατεβάσετε αυτά τα στοιχεία είναι να πάτε στη διεύθυνση http://www.coreservlets.com/, να ανοίξετε το Κεφάλαιο 4 (Chapter 4) της αρχειοθήκης πηγαίου κώδικα, και να αναζητήσετε τους άμεσους συνδέσμους για τα τρία αρχεία JAR.

Ο πιο φορητός τρόπος εγκατάστασης των στοιχείων είναι να ακολουθήσετε την καθιερωμένη προσέγγιση:

- Για ανάπτυξη, αναφέρετε τα αρχεία JAR στη μεταβλητή `CLASSPATH`.
- Για εκδίπλωση, τοποθετήστε τα αρχεία JAR στον κατάλογο WEB-INF/lib της εφαρμογής Ιστού.

Για το χειρισμό αρχείων JAR όπως αυτά εδώ — τα οποία χρησιμοποιούνται από πολλές εφαρμογές Ιστού — πολλοί προγραμματιστές χρησιμοποιούν ειδικές λειτουργίες του διακομιστή που υποστηρίζουν την κοινοχρησία αρχείων JAR μεταξύ πολλών εφαρμογών Ιστού. Για παράδειγμα, ο Tomcat επιτρέπει την τοποθέτηση των κοινών αρχείων JAR στον κατάλογο *κατ_εγκατάστασης_tomcat*/common/lib. Μια άλλη λύση, την οποία χρησιμοποιούν πολλοί προγραμματιστές στον υπολογιστή όπου κάνουν ανάπτυξη εφαρμογών, είναι η τοποθέτηση των τριών αρχείων JAR στον κατάλογο *κατ_εγκατάστασης_sdk*/jre/lib/ext. Με αυτόν τον τρόπο τα αρχεία JAR είναι αυτόματα προσπελάσιμα και από το περιβάλλον ανάπτυξης και από τον τοπικά εγκατεστημένο διακομιστή. Και οι δύο αυτές μέθοδοι αποτελούν χρήσιμες λύσεις εφόσον θυμάστε ότι ο κατάλογος *η_εφαρμογή_ιστού_σας*/WEB-INF/lib είναι η μόνη καθιερωμένη θέση για το διακομιστή εκδίπλωσης.

4.8 Επανεμφάνιση της φόρμας εισόδου για διόρθωση παραμέτρων

Μερικές φορές είναι εύλογο να χρησιμοποιούμε προεπιλεγμένες τιμές όταν ο χρήστης έχει παραλείψει να συμπληρώσει ορισμένα πεδία της φόρμας. Άλλες φορές, όμως, δεν υπάρχουν εύλογες προεπιλεγμένες τιμές για να χρησιμοποιήσουμε, και έτσι θα πρέπει να επανεμφανίσουμε τη φόρμα στο χρήστη. Υπάρχουν δύο επιθυμητές ιδιότητες οι οποίες κάνουν αδύνατη τη χρήση μιας κανονικής φόρμας HTML για αυτό το σενάριο:

- Οι χρήστες δεν θα πρέπει να ξαναπληκτρολογήσουν τιμές που έχουν ήδη εισαγάγει.
- Θα πρέπει να επισημαίνονται με εμφανή τρόπο τα πεδία της φόρμας που δεν έχουν συμπληρωθεί.

Επιλογές επανεμφάνισης

Τι μπορούμε λοιπόν να κάνουμε για να υλοποιήσουμε αυτές τις ιδιότητες; Η πλήρης περιγραφή των δυνατών προσεγγίσεων είναι κάπως περίπλοκη και απαιτεί γνώσεις αρκετών τεχνικών θεμάτων (π.χ. Struts, JSTL) που δεν καλύπτονται σε αυτό το βιβλίο. Έτσι θα πρέπει να ανατρέξετε στο βιβλίο More Servlets and JavaServer Pages για λεπτομέρειες· εδώ θα παρουσιάσουμε απλώς μια σύντομη επισκόπηση.

- **Η ίδια μικροϋπηρεσία παρουσιάζει τη φόρμα, επεξεργάζεται τα δεδομένα, και παρουσιάζει τα αποτελέσματα.** Η μικροϋπηρεσία πρώτα εξετάζει τα εισερχόμενα δεδομένα αίτησης: αν δεν βρει τίποτε, παρουσιάζει μια κενή φόρμα. Αν η μικροϋπηρεσία βρει κάποιο τμήμα των δεδομένων αίτησης, τότε τα εξάγει, τα επανατοποθετεί στη φόρμα, και σημειώνει τα υπόλοιπα πεδία που λείπουν. Αν η μικροϋπηρεσία βρει όλα τα δεδομένα που απαιτούνται, επεξεργάζεται την αίτηση και εμφανίζει τα αποτελέσματα. Σε αυτή τη φόρμα είναι κενή η ιδιότητα `ACTION`, έτσι ώστε οι υποβολές της φόρμας να πηγαίνουν αυτόματα στο ίδιο URL με αυτό της φόρμας. Αυτή είναι η μοναδική προσέγγιση για την οποία έχουμε ήδη περιγράψει όλες τις απαραίτητες τεχνικές, οπότε είναι και η προσέγγιση που θα παρουσιάσουμε σε αυτή την ενότητα.
- **Μία μικροϋπηρεσία παρουσιάζει τη φόρμα· μια δεύτερη μικροϋπηρεσία επεξεργάζεται τα δεδομένα και παρουσιάζει τα αποτελέσματα.** Αυτή η επιλογή είναι καλύτερη από την πρώτη, αφού διαχωρίζει τις δύο εργασίες και διατηρεί κάθε μικροϋπηρεσία πιο μικρή και πιο εύκολη στη διαχείριση. Όμως η χρήση αυτής της προσέγγισης απαιτεί δύο τεχνικές τις οποίες δεν έχουμε περιγράψει μέχρι τώρα: πώς γίνεται η μεταφορά του ελέγχου από τη μία μικροϋπηρεσία στην άλλη, και πώς προσπελάζονται από μια μικροϋπηρεσία ειδικά δεδομένα του χρήστη τα οποία έχουν δημιουργηθεί από κάποια άλλη μικροϋπηρεσία. Η μεταφορά από τη μία μικροϋπηρεσία στην άλλη μπορεί να πραγματοποιηθεί με τη μέθοδο `response.sendRedirect` (δείτε το Κεφάλαιο 6, "Δημιουργία απάντησης διακομιστή: Κωδικοί κατάστασης HTTP") ή με τη μέθοδο `forward` της κλάσης `RequestDispatcher` (δείτε το Κεφάλαιο 15, "Ολοκλήρωση μικροϋπηρεσιών και σελίδων JSP: Η αρχιτεκτονική Model View Controller (MVC)"). Ο ευκολότερος τρόπος για τη μεταβίβαση δεδομένων από τη μικροϋπηρεσία επεξεργασίας στη μικροϋπηρεσία που εμφανίζει τη φόρμα είναι η αποθήκευσή τους στο αντικείμενο `HttpSession` (δείτε το Κεφάλαιο 9, "Παρακολούθηση συνεδρίας").
- **Μια σελίδα JSP παρουσιάζει "χειρονακτικά" τη φόρμα· μια μικροϋπηρεσία ή σελίδα JSP επεξεργάζεται τα δεδομένα και παρουσιάζει τα αποτελέσματα.** Αυτή είναι μια εξαιρετική επιλογή η οποία χρησιμοποιείται ευρέως. Ωστόσο, απαιτεί επιπρόσθετα γνώσεις για τις σελίδες JSP, εκτός από τις γνώσεις που αναφέρθηκαν στην προηγούμενη

παράγραφο (πώς γίνεται η μεταφορά από τη μικροϋπηρεσία επεξεργασίας δεδομένων στη σελίδα της φόρμας και πώς χρησιμοποιείται η παρακολούθηση συνεδρίας για την αποθήκευση ειδικών δεδομένων του χρήστη). Πιο συγκεκριμένα, θα πρέπει να γνωρίζετε πώς χρησιμοποιούνται οι παραστάσεις JSP (δείτε το Κεφάλαιο 11, "Κλήση κώδικα Java με στοιχεία σεναρίου JSP") ή η `jsp:getProperty` (δείτε το Κεφάλαιο 14, "Χρήση στοιχείων JavaBeans σε έγγραφα JSP") για την εξαγωγή μερικών δεδομένων από το αντικείμενο δεδομένων και τη μεταφορά τους σε μια φόρμα HTML.

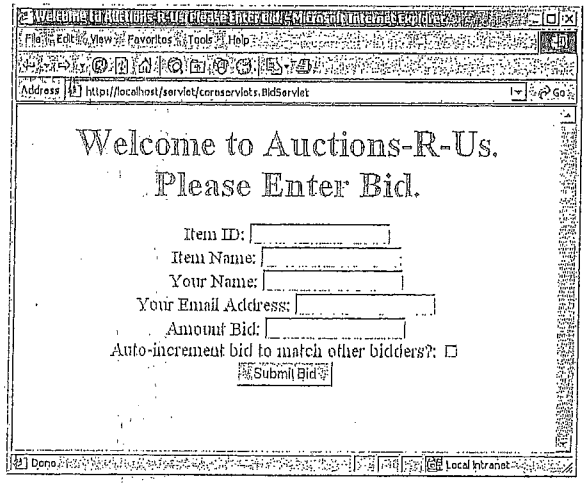
- **Μία σελίδα JSP παρουσιάζει τη φόρμα, συμπληρώνοντας αυτόματα τα πεδία με τιμές που λαμβάνονται από ένα αντικείμενο δεδομένων. Μια μικροϋπηρεσία ή σελίδα JSP επεξεργάζεται τα δεδομένα και παρουσιάζει τα αποτελέσματα.** Αυτή είναι ίσως η καλύτερη επιλογή από όλες. Εκτός όμως από τις τεχνικές που περιγράφονται στην προηγούμενη παράγραφο, απαιτεί προσαρμοσμένες ετικέτες JSP που μιμούνται στοιχεία μιας φόρμας HTML αλλά χρησιμοποιούν αυτόματα καθορισμένες τιμές. Μπορείτε είτε να γράψετε μόνοι σας αυτές τις ετικέτες, ή να χρησιμοποιήσετε έτοιμες ετικέτες όπως αυτές που περιλαμβάνονται με την JSTL ή το Apache Struts (περιγραφή των προσαρμοσμένων ετικετών, της JSTL, και του Struts μπορείτε να βρείτε στο βιβλίο More Servlets and JavaServer Pages).

Μικροϋπηρεσία που επεξεργάζεται προσφορές δημοπρασίας

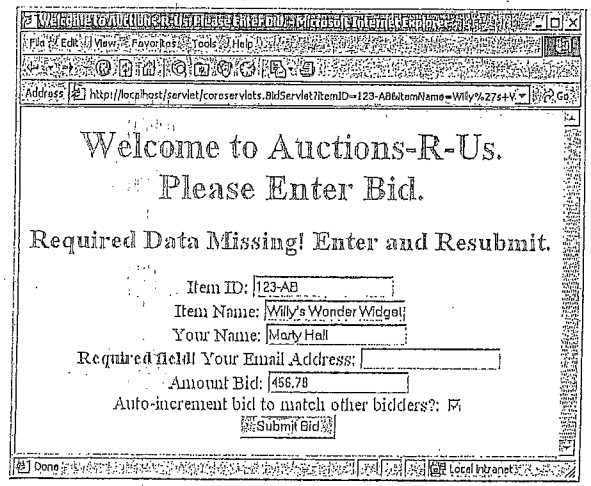
Για να δείξουμε την πρώτη από τις επιλογές επανεμφάνισης μιας φόρμας, ας εξετάσουμε μια μικροϋπηρεσία που επεξεργάζεται προσφορές σε μια τοποθεσία δημοπρασιών. Οι Εικόνες 4-14 μέχρι 4-16 δείχνουν την επιθυμητή έξοδο: η μικροϋπηρεσία αρχικά εμφανίζει μια κενή φόρμα, στην περίπτωση που έχει υποβληθεί μόνο ένα τμήμα των δεδομένων επανεμφανίζει τη φόρμα σημειώνοντας τα δεδομένα που λείπουν, και επεξεργάζεται την αίτηση όταν υποβληθούν πλήρως τα δεδομένα.

Για να επιτύχουμε αυτή τη συμπεριφορά, η μικροϋπηρεσία (Λίστα 4.17) εκτελεί τα ακόλουθα βήματα.

1. Συμπληρώνει ένα αντικείμενο `BidInfo` (Λίστα 4.17) με τα δεδομένα της αίτησης, χρησιμοποιώντας τη μέθοδο `BeanUtilities.populateBean` (δείτε την Ενότητα 4.7, "Αυτόματη συμπλήρωση αντικειμένων Java: Κόκκοι φόρμας"), έτσι ώστε να "ταιριάξει" αυτόματα τα ονόματα των παραμέτρων της αίτησης με τις ιδιότητες του κόκκου και να εκτελέσει τις απλές μετατροπές τύπου δεδομένων.
2. Ελέγχει αν το αντικείμενο `BidInfo` είναι εντελώς κενό (κανένα πεδίο δεν άλλαξε από τις προεπιλεγμένες τιμές του). Αν ισχύει κάτι τέτοιο, καλείται η `showEntryForm` για να εμφανιστεί η αρχική φόρμα εισαγωγής δεδομένων.

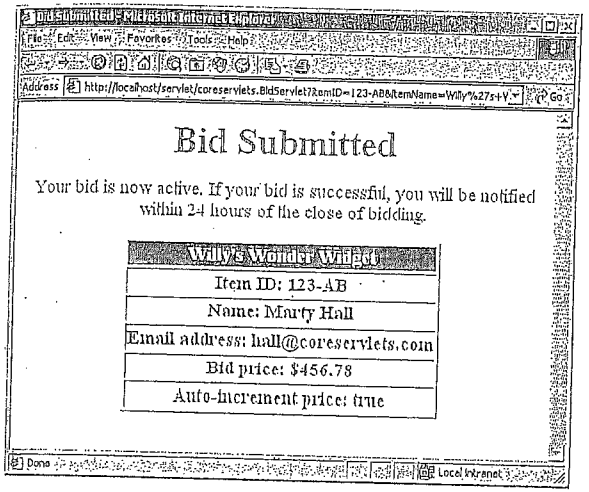


Εικόνα 4-14 Αρχική μορφή της μικροϋπηρεσίας: παρουσιάζει μια φόρμα για τη συλλογή δεδομένων για μια προσφορά σε κάποια δημοπρασία.



Εικόνα 4-15 Η μικροϋπηρεσία με ημιτελή δεδομένα. Αν ο χρήστης υποβάλει μια φόρμα η οποία δεν έχει συμπληρωθεί πλήρως, η μικροϋπηρεσία επανεμφανίζει τη φόρμα. Τα προηγούμενα δεδομένα του χρήστη διατηρούνται και επισημαίνονται τα δεδομένα που λείπουν.

3. Ελέγχει αν το αντικείμενο `BidInfo` είναι μερικώς κενό (μερικά, αλλά όχι όλα τα πεδία, έχουν αλλάξει από τις προεπιλεγμένες τιμές τους). Αν ισχύει κάτι τέτοιο, τότε καλείται η `showEntryForm` για να εμφανίσει τη φόρμα εισαγωγής με ένα προειδοποιητικό μήνυμα και με επισημασμένες τις τιμές που λείπουν. Τα πεδία στα οποία ο χρήστης έχει ήδη εισαγάγει τιμές διατηρούν τις προηγούμενες τιμές τους.
4. Ελέγχει αν το αντικείμενο `BidInfo` έχει συμπληρωθεί πλήρως. Σε αυτή την περίπτωση, καλεί τη `showBid` για να επεξεργαστεί τα δεδομένα και να εμφανίσει τα αποτελέσματα.



Εικόνα 4-16 Η μικροϋπηρεσία όταν έχουν συμπληρωθεί όλα τα δεδομένα: παρουσιάζει τα αποτελέσματα.

Λίστα 4.16 BidServlet.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import coreservlets.beans.*;

/** Παράδειγμα απλοποιημένης φόρμας επεξεργασίας.
 * Δείχνει δύο λειτουργίες:
 *  <OL>
 *    <LI>Αυτόματη συμπλήρωση ενός κόκκου με βάση τις
 *        παραμέτρους της εισερχόμενης αίτησης.
```

Λίστα 4.16 BidServlet.java *(συνέχεια)*

```
 *    <LI>Χρήση της ίδιας μικροϋπηρεσίας τόσο για την παραγωγή
 *        της φόρμας εισαγωγής, όσο και για την επεξεργασία
 *        των αποτελεσμάτων. Έτσι, όταν παραλείπονται πεδία,
 *        η μικροϋπηρεσία μπορεί να επανεμφανίσει τη φόρμα χωρίς
 *        να χρειάζεται ο χρήστης να εισαγάγει πάλι τις τιμές που
 *        έχει πληκτρολογήσει ήδη.
 *  </UL>
public class BidServlet extends HttpServlet {

  /** Προσπάθεια συμπλήρωσης ενός κόκκου που αντιπροσωπεύει
   *  τις πληροφορίες της φόρμας δεδομένων την οποία έστειλε
   *  ο χρήστης. Αν τα δεδομένα είναι πλήρη, εμφανίζει
   *  τα αποτελέσματα. Αν η φόρμα δεν έχει καθόλου δεδομένα
   *  ή αν τα δεδομένα δεν είναι πλήρη, εμφανίζει ξανά τη φόρμα HTML
   *  που συλλέγει τα δεδομένα.
   */

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    BidInfo bid = new BidInfo();
    BeanUtilities.populateBean(bid, request);
    if (bid.isComplete()) {
      // Υπάρχουν όλα τα αναγκαία πεδία φόρμας: εμφάνιση αποτελέσματος.
      showBid(request, response, bid);
    } else {
      // Ελλιπή ή ημιτελή δεδομένα φόρμας: επανεμφάνιση φόρμας.
      showEntryForm(request, response, bid);
    }
  }

  /** Υπάρχουν όλα τα απαιτούμενα δεδομένα: εμφάνιση της σελίδας
   *  αποτελεσμάτων.
   */

  private void showBid(HttpServletRequest request,
                       HttpServletResponse response,
                       BidInfo bid)
      throws ServletException, IOException {
    submitBid(bid);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Bid Submitted";
    out.println
      (DOCTYPE +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\"><CENTER>\n" +
```


Λίστα 4.16 BidServlet.java *(συνέχεια)*

```
       "<H1>" + title + "</H1>\n" +
       "Your bid is now active. If your bid is successful,\n" +
       "you will be notified within 24 hours of the close\n" +
       "of bidding.\n" +
       "<P>\n" +
       "<TABLE BORDER=1>\n" +
       "  <TR><TH BGCOLOR=\"BLACK\"><FONT COLOR=\"WHITE\">" +
       bid.getItemName() + "</FONT>\n" +
       "  <TR><TH>Item ID: " +
       bid.getItemID() + "\n" +
       "  <TR><TH>Name: " +
       bid.getBidderName() + "\n" +
       "  <TR><TH>Email address: " +
       bid.getEmailAddress() + "\n" +
       "  <TR><TH>Bid price: $" +
       bid.getBidPrice() + "\n" +
       "  <TR><TH>Auto-increment price: " +
       bid.isAutoIncrement() + "\n" +
       "</TABLE></CENTER></BODY></HTML>");
  }

  /** Αν λείπουν εντελώς τα απαιτούμενα δεδομένα, εμφάνιση
   *  μιας κενής φόρμας. Αν τα δεδομένα λείπουν εν μέρει,
   *  προειδοποίηση του χρήστη, συμπλήρωση των πεδίων που έχουν
   *  ήδη τιμές, και προτροπή του χρήστη για τα πεδία που λείπουν.
   */

  private void showEntryForm(HttpServletRequest request,
                             HttpServletResponse response,
                             BidInfo bid)
      throws ServletException, IOException {
    boolean isPartlyComplete = bid.isPartlyComplete();
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title =
      "Welcome to Auctions-R-Us. Please Enter Bid.";
    out.println
      (DOCTYPE +
       "<HTML>\n" +
       "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
       "<BODY BGCOLOR=\"#FDF5E6\"><CENTER>\n" +
       "<H1>" + title + "</H1>\n" +
       warning(isPartlyComplete) +
       "<FORM>\n" +
       inputElement("Item ID", "itemID",
                    bid.getItemID(), isPartlyComplete) +
       inputElement("Item Name", "itemName",
                    bid.getItemName(), isPartlyComplete) +
       inputElement("Your Name", "bidderName",
```

Λίστα 4.16 BidServlet.java *(συνέχεια)*

```
                bid.getBidderName(), isPartlyComplete) +
    inputElement("Your Email Address", "emailAddress",
                bid.getEmailAddress(), isPartlyComplete) +
    inputElement("Amount Bid", "bidPrice",
                bid.getBidPrice(), isPartlyComplete) +
    checkbox("Auto-increment bid to match other bidders?",
             "autoIncrement", bid.isAutoIncrement()) +
    "<INPUT TYPE=\"SUBMIT\" VALUE=\"Submit Bid\">\n" +
    "</CENTER></BODY></HTML>");
}

private void submitBid(BidInfo bid) {
  // Κώδικας ειδικά για την εφαρμογή για την καταγραφή της
  // προσφοράς. Η ουσία είναι ότι μεταβιβάζετε ένα πραγματικό
  // αντικείμενο με ιδιότητες που έχουν συμπληρωθεί, και όχι μια
  // σειρά αλφαριθμητικών.
}

private String warning(boolean isFormPartlyComplete) {
  if(isFormPartlyComplete) {
    return("<H2>Required Data Missing! " +
           "Enter and Resubmit.</H2>\n");
  } else {
    return("");
  }
}

/** Δημιουργία ενός πεδίου κειμένου για είσοδο, με μια προτροπή στην
 *  αρχή. Αν το συγκεκριμένο πεδίο κειμένου δεν έχει τιμή αλλά τα
 *  υπόλοιπα πεδία έχουν τιμές (δηλαδή έχει υποβληθεί μια μερικώς
 *  συμπληρωμένη φόρμα), τότε εμφάνιση προειδοποίησης προς το χρήστη
 *  η οποία τον ενημερώνει ότι το πεδίο κειμένου είναι απαραίτητο.
 */

private String inputElement(String prompt,
                            String name,
                            String value,
                            boolean shouldPrompt) {
  String message = "";
  if (shouldPrompt && ((value == null) || value.equals(""))) {
    message = "<B>Required field!</B> ";
  }
  return(message + prompt + ": " +
         "<INPUT TYPE=\"TEXT\" NAME=\"" + name + "\"" +
         " VALUE=\"" + value + "\"><BR>\n");
}

private String inputElement(String prompt,
                            String name,
                            double value,
```


Λίστα 4.16 BidServlet.java *(συνέχεια)*

```
                            boolean shouldPrompt) {
  String num;
  if (value == 0.0) {
    num = "";
  } else {
    num = String.valueOf(value);
  }
  return(inputElement(prompt, name, num, shouldPrompt));
}

private String checkbox(String prompt,
                        String name,
                        boolean isChecked) {
  String result =
    prompt + ": " +
    "<INPUT TYPE=\"CHECKBOX\" NAME=\"" + name + "\"";
  if (isChecked) {
    result = result + " CHECKED";
  }
  result = result + "><BR>\n";
  return(result);
}

private final String DOCTYPE =
  "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
  "Transitional//EN\">\n";
}
```

Λίστα 4.17 BidInfo.java

```
package coreservlets.beans;

import coreservlets.*;

/** Κόκκος που αντιπροσωπεύει πληροφορίες για μια προσφορά
 *  σε τοποθεσία ιστού δημοπρασιών. Χρησιμοποιείται για επίδειξη
 *  της επανεμφάνισης μιας φόρμας που έχει ελλιπή δεδομένα.
*/

public class BidInfo {
  private String itemID = "";
  private String itemName = "";
  private String bidderName = "";
  private String emailAddress = "";
  private double bidPrice = 0;
  private boolean autoIncrement = false;
```

Λίστα 4.17 BidInfo.java *(συνέχεια)*

```
public String getItemName() {
  return(itemName);
}

public void setItemName(String itemName) {
  this.itemName = ServletUtilities.filter(itemName);
}

public String getItemID() {
  return(itemID);
}

public void setItemID(String itemID) {
  this.itemID = ServletUtilities.filter(itemID);
}

public String getBidderName() {
  return(bidderName);
}

public void setBidderName(String bidderName) {
  this.bidderName =  ServletUtilities.filter(bidderName);
}

public String getEmailAddress() {
  return(emailAddress);
}

public void setEmailAddress(String emailAddress) {
  this.emailAddress = ServletUtilities.filter(emailAddress);
}

public double getBidPrice() {
  return(bidPrice);
}

public void setBidPrice(double bidPrice) {
  this.bidPrice = bidPrice;
}

public boolean isAutoIncrement() {
  return(autoIncrement);
}

public void setAutoIncrement(boolean autoIncrement) {
  this.autoIncrement = autoIncrement;
}
```


Λίστα 4.17 BidInfo.java *(συνέχεια)*

```
/** Έχουν εισαχθεί όλα τα απαιτούμενα δεδομένα; Όλα τα άλλα εκτός
    του autoIncrement πρέπει να ορίζονται ρητά (η autoIncrement
    παίρνει εξ ορισμού την τιμή false).
*/

public boolean isComplete() {
  return(hasValue(getItemID()) &&
         hasValue(getItemName()) &&
         hasValue(getBidderName()) &&
         hasValue(getEmailAddress()) &&
         (getBidPrice() > 0));
}

/** Έχει εισαχθεί κάποιο από τα δεδομένα; */

public boolean isPartlyComplete() {
  boolean flag =
         (hasValue(getItemID()) ||
         hasValue(getItemName()) ||
         hasValue(getBidderName()) ||
         hasValue(getEmailAddress()) ||
         (getBidPrice() > 0) ||
         isAutoIncrement());
  return(flag);
}

private boolean hasValue(String val) {
  return((val != null) && (!val.equals("")));
}
}
```