ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΣΥΝΕΔΡΙΑΣ

Κεφάλαιο 9

Θέματα σε αυτό το Κεφάλαιο

- Υλοποίηση της παρακολούθησης συνεδρίας από το μηδέν
- Χρήση βασικής παρακολούθησης συνεδρίας
- Κατανόηση της API παρακολούθηση συνεδριών
- Διαφοροποίηση μεταξύ συνεδρίας διακομιστή και φυλλομετρητή
- Κωδικοποίηση URL
- Σύγκριση αποθήκευσης αμετάβλητων αντικειμένων και αποθήκευσης μεταβλητών αντικειμένων
- Παρακολούθηση του αριθμού επισκέψεων χρηστών
- Συγκέντρωση αγορών χρηστών
- Υλοποίηση ενός καλαθιού αγορών
- Δόμηση ενός online καταστήματος

Αυτό το κεφάλαιο αποτελεί μια εισαγωγική παρουσίαση της API μικροϋπηρεσιών για την παρακολούθηση συνεδριών (session-tracking API), όπου παρακολουθούνται δεδομένα ειδικά για κάθε χρήστη καθώς οι επισκέπτες κινούνται μέσα στην τοποθεσία σας.

9.1 Η ανάγκη της παρακολούθησης συνεδριών

Το HTTP είναι ένα πρωτόκολλο "χωρίς καταστάσεις" (stateless protocol): κάθε φορά που κάποιος πελάτης ανακτά μια ιστοσελίδα, ο πελάτης ανοίγει μία ξεχωριστή σύνδεση με το διακομιστή Ιστού και ο διακομιστής δεν διατηρεί αυτόματα πληροφορίες σχετικά με τον πελάτη. Ακόμα και στους διακομιστές που υποστηρίζουν διατηρούμενες (keep-alive) συνδέσεις HTTP και κρατούν τις υποδοχές (sockets) ανοικτές για πολλαπλές αιτήσεις πελάτη που πραγματοποιούνται διαδοχικά, δεν υπάρχει ενσωματωμένη υποστήριξη για τη διατήρηση των σχετικών πληροφοριών. Αυτή η έλλειψη προκαλεί μια σειρά από δυσκολίες. Για παράδειγμα, όταν κάποιοι πελάτες βρίσκονται σε ένα online κατάστημα και προσθέτουν ένα είδος στο καλάθι αγορών τους, πώς θα γνωρίζει ο διακομιστής τι υπάρχει ήδη μέσα στο καλάθι; Παρομοίως, όταν οι πελάτες αποφασίσουν να τερματίσουν τις αγορές τους, πώς θα μπορεί ο διακομιστής να προσδιορίσει ποια από τα καλάθια αγορών που έχει δημιουργήσει είναι δικά τους; Αν και αυτές οι ερωτήσεις φαίνονται απλές, λόγω των ανεπαρκειών του πρωτοκόλλου HTTP η απάντησή τους είναι εξαιρετικά πολύπλοκη.

Σε αυτό το πρόβλημα υπάρχουν τρεις τυπικές λύσεις: τα μπισκότα, η επανεγγραφή του URL, και τα κρυφά πεδία φόρμας. Στις υποενότητες που ακολουθούν αναφέρουμε συνοπτικά όλα εκείνα που θα χρειαζόταν να κάνετε αν έπρεπε να υλοποιήσετε μόνοι σας την παρακολούθηση συνε-

δριών (χωρίς να χρησιμοποιήσετε την ενσωματωμένη API παρακολούθησης συνεδριών) για καθεμία από αυτές τις μεθόδους επίλυσης του προβλήματος.

Μπισκότα

Μπορείτε να χρησιμοποιήσετε μπισκότα για να αποθηκεύσετε το αναγνωριστικό (ID) μιας συνεδρίας αγορών· σε κάθε επόμενη σύνδεση μπορείτε να αναζητήσετε το αναγνωριστικό της τρέχουσας συνεδρίας (η οποία ονομάζεται και *περίοδος εργασίας*) και μετά να χρησιμοποιήσετε αυτό το αναγνωριστικό για να εξαγάγετε πληροφορίες σχετικά με τη διεργασία από έναν πίνακα αναζήτησης που υπάρχει στο διακομιστή. Έτσι θα υπάρχουν στην πραγματικότητα δύο πίνακες: ένας πίνακας που θα συσχετίζει τα αναγνωριστικά συνεδριών με τους πίνακες χρηστών, καθώς και ο πίνακας των χρηστών με τα αποθηκευμένα δεδομένα για κάθε χρήστη. Για παράδειγμα, για την αρχική αίτηση η μικροϋπηρεσία θα μπορούσε να κάνει κάτι ανάλογο με το παρακάτω:

```
String sessionID = makeUniqueString();
HashMap sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);
Cookie sessionCookie = new Cookie("JSESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

Κατόπιν, σε επόμενες αιτήσεις, ο διακομιστής θα μπορούσε να χρησιμοποιεί τον πίνακα κατακερματισμού (hash table) `globalTable` για να συσχετίσει το αναγνωριστικό συνεδρίας από το μπισκότο `JSESSIONID` με τον πίνακα κατακερματισμού `sessionInfo` που περιέχει τα δεδομένα του κάθε χρήστη.

Αυτή η χρήση των μπισκότων αποτελεί μια εξαιρετική λύση, και είναι η προσέγγιση που χρησιμοποιείται ευρέως για το χειρισμό συνεδριών. Παρόλα αυτά, είναι εξαιρετικά βολικό το ότι οι μικροϋπηρεσίες διαθέτουν μια API υψηλότερου επιπέδου που χειρίζεται όλα αυτά τα θέματα, μαζί με τις παρακάτω επίπονες εργασίες:

- Εξαγωγή του μπισκότου που αποθηκεύει το αναγνωριστικό συνεδρίας από τα άλλα μπισκότα (άλλωστε, μπορεί να υπάρχουν πολλά μπισκότα).
- Προσδιορισμός των ανενεργών συνεδριών που έχουν λήξει και ανάκτησή τους.
- Συσχετισμός των πινάκων κατακερματισμού με κάθε αίτηση.
- Παραγωγή των μοναδικών αναγνωριστικών συνεδριών.

Επανεγγραφή των URL

Με αυτή την προσέγγιση ο πελάτης προσαρτά στο τέλος του κάθε URL κάποια πρόσθετα δεδομένα. Τα δεδομένα προσδιορίζουν τη συνεδρία, και ο διακομιστής συσχετίζει το αναγνωριστικό με τα αποθηκευμένα δεδομένα του χρήστη. Για παράδειγμα, με το URL http://host/path/file.html;jsessionid=a1234 το αναγνωριστικό συνεδρίας προσαρτάται ως jsessionid=a1234, οπότε το `a1234`

είναι το αναγνωριστικό που προσδιορίζει μοναδικά τον πίνακα ο οποίος σχετίζεται με το συγκεκριμένο χρήστη.

Η επανεγγραφή του URL αποτελεί μια μέτρια λύση για την παρακολούθηση συνεδριών, αν και έχει το πλεονέκτημα ότι λειτουργεί με φυλλομετρητές που δεν υποστηρίζουν μπισκότα ή στις περιπτώσεις που ο χρήσης τα έχει απενεργοποιήσει. Ωστόσο, αν υλοποιήσετε μόνοι σας την παρακολούθηση συνεδριών, η επανεγγραφή του URL έχει τα ίδια μειονεκτήματα με τα μπισκότα: με άλλα λόγια, το πρόγραμμα στο διακομιστή θα πρέπει να εκτελέσει μια απλή αλλά επίπονη και εκτενή επεξεργασία. Ακόμα και με τη χρήση μιας API υψηλού επιπέδου που χειρίζεται για λογαριασμό σας τις περισσότερες λεπτομέρειες, θα πρέπει να είστε πολύ προσεκτικοί έτσι ώστε κάθε URL που αναφέρεται στην τοποθεσία σας και επιστρέφεται στο χρήστη (ακόμα και με έμμεσο τρόπο, όπως τα πεδία `Location` σε ανακατευθύνσεις του διακομιστή) θα έχει προσαρτημένες τις πρόσθετες πληροφορίες. Αυτός ο περιορισμός σημαίνει ότι δεν μπορείτε να έχετε στατικές σελίδες HTML στην τοποθεσία σας (ή τουλάχιστον σελίδες που θα διαθέτουν συνδέσμους σε δυναμικές σελίδες της τοποθεσίας σας). Έτσι κάθε σελίδα θα πρέπει να παράγεται δυναμικά, με μικροϋπηρεσίες ή JSP. Ακόμα και όταν όλες οι σελίδες παράγονται δυναμικά, αν ο χρήστης εγκαταλείψει τη συνεδρία και επιστρέψει μέσω κάποιου σελιδοδείκτη ή συνδέσμου οι πληροφορίες συνεδρίας μπορεί να χαθούν, επειδή ο αποθηκευμένος σύνδεσμος θα περιέχει λανθασμένες πληροφορίες αναγνώρισης.

Κρυφά πεδία φόρμας

Όπως θα δούμε στο Κεφάλαιο 19 (Δημιουργία και επεξεργασία φορμών HTML), οι φόρμες HTML μπορούν να έχουν μία καταχώριση που μοιάζει με την ακόλουθη:

```
<INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">
```

Αυτή η καταχώριση σημαίνει ότι, όταν υποβληθεί η φόρμα, το συγκεκριμένο όνομα και η τιμή θα συμπεριληφθούν αυτόματα στα δεδομένα `GET` ή `POST`. Αυτό το κρυφό πεδίο μπορεί να χρησιμοποιηθεί για την αποθήκευση πληροφοριών σχετικά με τη συνεδρία, αλλά έχει το μεγάλο μειονέκτημα ότι λειτουργεί μόνο όταν η κάθε σελίδα παράγεται δυναμικά μέσω της υποβολής μιας φόρμας. Επειδή το πάτημα σε έναν κανονικό σύνδεσμο υπερ-κειμένου (`<A HREF...>`) δεν ισοδυναμεί με υποβολή φόρμας, τα κρυφά πεδία φόρμας δεν μπορούν να υποστηρίξουν γενική παρακολούθηση συνεδριών, αλλά μόνο την παρακολούθηση μιας συγκεκριμένης σειράς ενεργειών — όπως η πληρωμή των αγορών σε ένα online κατάστημα.

Παρακολούθηση συνεδριών στις μικροϋπηρεσίες

Οι μικροϋπηρεσίες παρέχουν μια εξαιρετική λύση παρακολούθησης συνεδριών: την API `HttpSession`. Αυτή η διασύνδεση υψηλού επιπέδου δομείται πάνω στα μπισκότα ή την επανεγγραφή των διευθύνσεων URL. Όλοι οι διακομιστές θα πρέπει να υποστηρίζουν την παρακολούθηση συνεδριών με μπισκότα, και οι περισσότεροι έχουν μια ρύθμιση με την οποία μπορείτε να περάσετε καθολικά σε επανεγγραφή των URL.

Και στις δύο περιπτώσεις ο συγγραφέας των μικροϋπηρεσιών δεν χρειάζεται να ασχοληθεί με πολλές λεπτομέρειες υλοποίησης, δεν χρειάζεται να χειριστεί ρητά τα μπισκότα ή τις πληροφορίες που έχουν προσαρτηθεί στις διευθύνσεις URL, και του παρέχεται αυτόματα μια χρήσιμη θέση όπου μπορεί να αποθηκεύσει αντικείμενα τα οποία σχετίζονται με κάθε συνεδρία.

9.2 Βασικά στοιχεία της παρακολούθησης συνεδριών

Η χρήση των συνεδριών στις μικροϋπηρεσίες είναι απλή και περιλαμβάνει τέσσερα βασικά στάδια. Θα δούμε τώρα μια σύνοψή τους, και στη συνέχεια θα παρουσιάσουμε τις λεπτομέρειες.

1. **Προσπέλαση του αντικειμένου συνεδρίας που σχετίζεται με την τρέχουσα αίτηση.** Κλήση της μεθόδου `request.getSession` για τη λήψη ενός αντικειμένου `HttpSession`, το οποίο είναι ένας απλός πίνακας κατακερματισμού με δεδομένα ειδικά για το χρήστη.
2. **Αναζήτηση πληροφοριών που σχετίζονται με μια συνεδρία.** Κλήση της `getAttribute` για το αντικείμενο `HttpSession`, ρητή μετατροπή της επιστρεφόμενης τιμής στον κατάλληλο τύπο, και έλεγχος αν το αποτέλεσμα είναι `null`.
3. **Αποθήκευση πληροφοριών σε μια συνεδρία.** Χρήση της `setAttribute` με ένα κλειδί και μία τιμή.
4. **Απόρριψη δεδομένων συνεδρίας.** Κλήση της `removeAttribute` για την απόρριψη μιας συγκεκριμένης τιμής. Κλήση της `Invalidate` για την απόρριψη μιας ολόκληρης συνεδρίας. Κλήση της `logout` για την αποσύνδεση (log out) του πελάτη από το διακομιστή Ιστού και την ακύρωση όλων των συνεδριών που σχετίζονται με το χρήστη.

Προσπέλαση του αντικειμένου συνεδρίας που σχετίζεται με την τρέχουσα αίτηση

Τα αντικείμενα συνεδρίας είναι τύπου `HttpSession`, αλλά κατά βάση είναι πίνακες κατακερματισμού στους οποίους μπορούν να αποθηκευτούν διάφορα αντικείμενα χρήστη (όπου το κάθε αντικείμενο σχετίζεται με κάποιο κλειδί). Η αναζήτηση του αντικειμένου `HttpSession` γίνεται με κλήση της μεθόδου `getSession` της κλάσης `HttpServletRequest`, με τον εξής τρόπο:

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

Στο παρασκήνιο, το σύστημα εξάγει ένα αναγνωριστικό χρήστη (user ID) από το μπισκότο ή τα δεδομένα που έχουν προσαρτηθεί στο URL, και στη συνέχεια χρησιμοποιεί το αναγνωριστικό ως κλειδί σε έναν πίνακα με αντικείμενα `HttpSession` τα οποία έχουν δημιουργηθεί στο παρελθόν. Όμως αυτό γίνεται χωρίς τη μεσολάβηση του προγραμματιστή: εσείς απλώς καλείτε τη μέθοδο `getSession`. Αν δεν βρεθεί αναγνωριστικό συνεδρίας στο εισερχόμενο μπισκότο ή στις πληροφορίες που έχουν προσαρτηθεί στο URL, το σύστημα δημιουργεί ένα νέο, κενό αντικείμενο συνεδρίας. Επιπρόσθετα, όταν χρησιμοποιούνται μπισκότα (που είναι η προεπιλογή), το σύστημα δημιουργεί επίσης ένα εξερχόμενο μπισκότο με το όνομα `JSESSIONID` και με μια μοναδική τιμή που αντιπροσωπεύει το αναγνωριστικό της συνεδρίας. Έτσι, αν και καλείτε τη μέθοδο `getSession` για την *αίτηση*, η κλήση μπορεί να επηρεάσει την *απάντηση*. Επομένως, επιτρέπεται να καλέσετε τη `request.getSession` μόνον όταν είναι έγκυρος ο ορισμός κεφαλίδων απάντησης HTTP: δηλαδή πριν από την αποστολή περιεχομένου εγγράφου στον πελάτη.

Η προσέγγιση του βιβλίου

Καλέστε τη `request.Session` πριν αποστείλετε οποιοδήποτε περιεχόμενο εγγράφου στον πελάτη.

Τώρα, αν σχεδιάζετε να προσθέσετε δεδομένα στη συνεδρία ανεξάρτητα από το αν υπάρχουν ήδη δεδομένα σε αυτήν, η μέθοδος `getSession()` (ή, ισοδύναμα, η μέθοδος `getSession(true)`) είναι η κατάλληλη μέθοδος, επειδή δημιουργεί μια νέα συνεδρία αν αυτή δεν υπάρχει. Υποθέστε όμως ότι θέλετε απλώς να τυπώσετε τις πληροφορίες που υπάρχουν στη συνεδρία, όπως στην περίπτωση της σελίδας "View Cart" (Προβολή του καλαθιού αγορών) σε μια τοποθεσία ηλεκτρονικού εμπορίου. Σε αυτή την περίπτωση η δημιουργία μιας νέας συνεδρίας αποτελεί σπατάλη όταν αυτή δεν υπάρχει ήδη. Έτσι μπορείτε να χρησιμοποιήσετε τη μέθοδο `getSession(false)`, η οποία επιστρέφει τιμή `null` αν δεν υπάρχει συνεδρία για τον τρέχοντα πελάτη. Ας δούμε ένα παράδειγμα.

```
HttpSession session = request.getSession(false);
if (session == null) {
  printMessageSayingCartIsEmpty();
} else {
  extractCartAndPrintContents(session);
}
```

Αναζήτηση πληροφοριών που σχετίζονται με μια συνεδρία

Τα αντικείμενα `HttpSession` "ζουν" στο διακομιστή — δεν περιπλανώνται στο δίκτυο — και συσχετίζονται απλώς με τον πελάτη μέσω ενός παρασκηνιακού μηχανισμού, όπως είναι τα μπισκότα και η επανεγγραφή των διευθύνσεων URL. Αυτά τα αντικείμενα συνεδριών έχουν μια ενσωματωμένη δομή δεδομένων (έναν πίνακα κατακερματισμού) στον οποίο μπορείτε να αποθηκεύσετε οποιονδήποτε αριθμό κλειδιών και αντίστοιχων τιμών. Για να αναζητήσετε μια ήδη αποθηκευμένη τιμή χρησιμοποιείτε τη μέθοδο `session.getAttribute("key")`. Ο επιστρεφόμενος τύπος δεδομένων είναι `Object`, έτσι πρέπει να κάνετε ρητή μετατροπή (typecast) στο συγκεκριμένο τύπο δεδομένων που σχετίζεται με το όνομα της ιδιότητας στη συνεδρία. Αν δεν υπάρχει τέτοια ιδιότητα η επιστρεφόμενη τιμή είναι `null`, οπότε θα πρέπει να ελέγχετε για την παρουσία της τιμής `null` πριν καλέσετε μεθόδους για αντικείμενα που σχετίζονται με συνεδρίες.

Ας δούμε ένα αντιπροσωπευτικό παράδειγμα:

```
HttpSession session = request.getSession();
SomeClass value =
  (SomeClass)session.getAttribute("someIdentifier");
if (value = null) {//Δεν υπάρχει αυτό το αντικείμενο στη συνεδρία
  value = new SomeClass(...);
  session.setAttribute("someIdentifier", value);
}
doSomethingWith(value);
```

Στις περισσότερες περιπτώσεις θα έχετε στο μυαλό σας ένα συγκεκριμένο όνομα ιδιότητας και θα θέλετε να βρείτε την τιμή (αν υπάρχει) που σχετίζεται με αυτό το όνομα. Ωστόσο, μπορείτε να βρείτε όλα τα ονόματα των ιδιοτήτων που σχετίζονται με μια συγκεκριμένη συνεδρία καλώντας τη μέθοδο `getAttributeNames`, η οποία επιστρέφει μια απαρίθμηση (`Enumeration`).

Συσχετισμός πληροφοριών με μια συνεδρία

Όπως αναφέραμε στην προηγούμενη υποενότητα, για να *διαβάσετε* τις πληροφορίες που σχετίζονται με μια συνεδρία χρησιμοποιείτε τη μέθοδο `getAttribute`. Για να *καθορίσετε* πληροφορίες χρησιμοποιείτε τη μέθοδο `setAttribute`. Για να επιτρέψετε στις τιμές σας να έχουν παράπλευρα αποτελέσματα όταν αποθηκεύονται σε μια συνεδρία, θα πρέπει το αντικείμενο που συσχετίζετε με τη συνεδρία να υλοποιεί τη διασύνδεση `HttpSessionBindingListener`. Με αυτόν τον τρόπο, κάθε φορά που καλείται η μέθοδος `setAttribute` για κάποιο από αυτά τα αντικείμενα, θα καλείται αυτόματα η μέθοδος `valueBound` αμέσως μετά από αυτή.

Θα πρέπει να θυμάστε ότι η μέθοδος `setAttribute` αντικαθιστά τυχόν προηγούμενες τιμές· για να αφαιρέσετε κάποια τιμή χωρίς να αποδώσετε μια καινούργια, θα πρέπει να χρησιμοποιήσετε τη μέθοδο `removeAttribute`. Η μέθοδος αυτή ενεργοποιεί τη μέθοδο `valueUnbound` για τις τιμές που υλοποιούν τη διασύνδεση `HttpSessionBindingListener`.

Το επόμενο απόσπασμα κώδικα είναι ένα παράδειγμα προσθήκης πληροφοριών σε μια συνεδρία. Η προσθήκη πληροφοριών μπορεί να γίνει με δύο τρόπους: μέσω της προσθήκης μιας νέας ιδιότητας συνεδρίας (όπως στη γραμμή του παραδείγματος που είναι με έντονα γράμματα) ή με την επαύξηση ενός αντικείμενου που ήδη βρίσκεται στη συνεδρία (όπως στην τελευταία γραμμή του παραδείγματος). Αυτή η διάκριση αναπτύσσεται περαιτέρω στα παραδείγματα των Ενοτήτων 9.7 και 9.8, όπου θα δούμε τη διαφορά ανάμεσα στη χρήση αμετάβλητων (immutable) και μεταβλητών (mutable) αντικειμένων ως ιδιοτήτων συνεδριών.

```
HttpSession session = request.getSession();
SomeClass value =
  (SomeClass)session.getAttribute("someIdentifier");
if (value = null) {//Δεν υπάρχει αυτό το αντικείμενο στη συνεδρία
  value = new SomeClass(...);
  session.setAttribute("someIdentifier", value);
}
doSomethingWith(value);
```


Γενικά, οι ιδιότητες συνεδριών πρέπει απλώς να έχουν τύπο δεδομένων `Object` (δηλαδή, μπορεί να είναι οτιδήποτε άλλο εκτός από `null` ή κάποιος θεμελιώδης τύπος δεδομένων, όπως `int`, `double`, ή `boolean`). Ορισμένοι διακομιστές εφαρμογών υποστηρίζουν, όμως, κατανεμημένες εφαρμογές Ιστού, όπου μια εφαρμογή διαμοιράζεται σε μια συστοιχία (cluster) φυσικών διακομιστών. Η παρακολούθηση συνεδριών θα πρέπει να εξακολουθεί να λειτουργεί και σε τέτοιες περιπτώσεις, οπότε το σύστημα θα πρέπει να είναι σε θέση να μετακινεί αντικείμενα συνεδριών από μηχάνημα σε μηχάνημα. Έτσι, αν δουλεύετε σε ένα τέτοιο περιβάλλον και έχετε χαρακτηρίσει την εφαρμογή σας ως κατανεμημένη, θα πρέπει να ικανοποιήσετε την επιπλέον απαίτηση να υλοποιούν οι ιδιότητες συνεδριών τη διασύνδεση `Serializable`.

Απόρριψη δεδομένων συνεδρίας

Όταν τελειώσετε με τα δεδομένα συνεδρίας ενός χρήστη, έχετε τρεις επιλογές:

- **Αφαίρεση μόνο των δεδομένων που δημιούργησε η μικροϋπηρεσία σας.** Μπορείτε να καλέσετε τη μέθοδο `removeAttribute("key")` για να απορρίψετε την τιμή που σχετίζεται με το συγκεκριμένο κλειδί. Αυτή είναι η πιο συνηθισμένη προσέγγιση.
- **Διαγραφή ολόκληρης της συνεδρίας (στην τρέχουσα εφαρμογή Ιστού).** Μπορείτε να καλέσετε τη μέθοδο `invalidate` για να απορρίψετε μια ολόκληρη συνεδρία. Θα πρέπει απλώς να θυμάστε ότι αυτό προκαλεί την απώλεια όλων των δεδομένων συνεδρίας του χρήστη, και όχι μόνο των δεδομένων που έχει δημιουργήσει η μικροϋπηρεσία σας ή η σελίδα JSP. Έτσι *όλες* οι μικροϋπηρεσίες και οι σελίδες JSP μιας εφαρμογής Ιστού θα πρέπει να συμφωνήσουν για τις περιπτώσεις στις οποίες μπορεί να κληθεί η μέθοδος `invalidate`.
- **Αποσύνδεση του χρήστη και διαγραφή όλων των συνεδριών που ανήκουν σε αυτόν.** Τέλος, στους διακομιστές που υποστηρίζουν μικροϋπηρεσίες 2.4 και JSP 2.0, μπορείτε να καλέσετε τη μέθοδο `logout` για να αποσυνδέσετε το χρήστη από το διακομιστή Ιστού και να ακυρώσετε (invalidate) όλες τις συνεδρίες (το πολύ μία για κάθε εφαρμογή Ιστού) που σχετίζονται με αυτόν το χρήστη. Και πάλι, επειδή αυτή η ενέργεια επηρεάζει και άλλες μικροϋπηρεσίες εκτός από τη δική σας, φροντίστε να συντονίσετε τη χρήση της διαταγής `logout` με τους άλλους προγραμματιστές της τοποθεσίας σας.

9.3 Η API παρακολούθησης συνεδριών

Αν και οι ιδιότητες συνεδριών (δηλαδή τα δεδομένα του χρήστη) είναι τα τμήματα πληροφορίας συνεδριών που κυρίως σας ενδιαφέρουν, υπάρχουν και άλλες πληροφορίες που μερικές φορές θα σας φανούν χρήσιμες. Η ακόλουθη είναι μια σύνοψη των διαθέσιμων μεθόδων της κλάσης `HttpSession`.

public Object getAttribute(String name)

Αυτή η μέθοδος εξάγει μια αποθηκευμένη τιμή από ένα αντικείμενο συνεδρίας. Επιστρέφει τιμή `null` αν καμία τιμή δεν σχετίζεται με το συγκεκριμένο όνομα.

public Enumeration getAttributeNames()

Αυτή η μέθοδος επιστρέφει τα ονόματα όλων των ιδιοτήτων στη συνεδρία.

public void setAttribute(String name, Object value)

Αυτή η μέθοδος συσχετίζει μια τιμή με ένα όνομα. Αν το αντικείμενο που παρέχεται στη μέθοδο `setAttribute` υλοποιεί τη διασύνδεση `HttpSessionBindingListener`, τότε καλείται η μέθοδος `valueBound` του αντικειμένου μετά από την αποθήκευσή του στη συνεδρία. Παρομοίως, αν η προηγούμενη τιμή υλοποιεί τη διασύνδεση `HttpSessionBindingListener`, καλείται η μέθοδος `valueUnbound`.

public void removeAttribute(String name)

Αυτή η μέθοδος αφαιρεί τις τιμές που σχετίζονται με το συγκεκριμένο όνομα. Αν η τιμή που αφαιρείται υλοποιεί τη διασύνδεση `HttpSessionBindingListener`, τότε καλείται η μέθοδος `valueUnbound`.

public void invalidate()

Αυτή η μέθοδος ακυρώνει τη συνεδρία και αποσυνδέει όλα τα αντικείμενα που σχετίζονται με αυτή. Να χρησιμοποιείτε αυτή τη μέθοδο με προσοχή· θυμηθείτε ότι οι συνεδρίες σχετίζονται με χρήστες (δηλαδή πελάτες), και όχι με μεμονωμένες μικροϋπηρεσίες ή σελίδες JSP. Έτσι, αν ακυρώσετε μία συνεδρία, μπορεί να καταστρέψετε δεδομένα τα οποία χρησιμοποιεί κάποια άλλη μικροϋπηρεσία ή σελίδα JSP.

public void logout()

Αυτή η μέθοδος αποσυνδέει (logout) το χρήστη από το διακομιστή Ιστού και ακυρώνει *όλες* τις συνεδρίες που σχετίζονται με το συγκεκριμένο πελάτη. Η εμβέλεια της αποσύνδεσης είναι ίδια με την εμβέλεια της πιστοποίησης ταυτότητας (authentication). Για παράδειγμα, αν ο διακομιστής υλοποιεί απλή σύνδεση (single sign-on), η κλήση της μεθόδου `logout` αποσυνδέει τον πελάτη από όλες τις εφαρμογές Ιστού στο διακομιστή και ακυρώνει όλες τις συνεδρίες (το πολύ μία για κάθε εφαρμογή Ιστού) που σχετίζονται με τον πελάτη. Για περισσότερες λεπτομέρειες, δείτε τα κεφάλαια σχετικά με την ασφάλεια εφαρμογών Ιστού στο βιβλίο More Servlets and JavaServer Pages.

public String getId()

Αυτή η μέθοδος επιστρέφει το μοναδικό αναγνωριστικό που παράγεται για κάθε συνεδρία. Είναι χρήσιμη για αποσφαλμάτωση ή σύνδεση, και σε σπάνιες περιπτώσεις, για την προγραμματιστική μεταφορά τιμών από τη μνήμη σε μια βάση δεδομένων (ωστόσο, ορισμένοι διακομιστές J2EE μπορούν να το κάνουν αυτό αυτόματα).

public boolean isNew()

Αυτή η μέθοδος επιστρέφει τιμή `true` αν ο πελάτης (φυλλομετρητής) δεν έχει δει ποτέ τη συνεδρία, συνήθως επειδή η συνεδρία μόλις δημιουργήθηκε και όχι επειδή γίνεται αναφορά σε αυτή από κάποια εισερχόμενη αίτηση πελάτη. Επιστρέφει τιμή `false` για προϋπάρχουσες συνεδρίες. Ο βασικός λόγος για τον οποίο αναφέρουμε αυτή τη μέθοδο είναι για να σας πούμε να την αποφύγετε: η μέθοδος `isNew` δεν είναι τόσο χρήσιμη όσο φαίνεται εκ πρώτης όψεως. Πολλοί αρχάριοι προγραμματιστές προσπαθούν να χρησιμοποιήσουν τη μέθοδο `isNew` για να προσδιορίσουν αν οι χρήστες έχουν χρησιμοποιήσει πάλι τις μικροϋπηρεσίες τους (μέχρι τη λήξη της συνεδρίας), γράφοντας κώδικα όπως ο ακόλουθος:

```
HttpSession session = request.getSession();
if (session.isNew()) {
  doStuffFonNewbies();
} else {
  doStuffForReturnVisitors();  //Λάθος
}
```

Λάθος! Πράγματι, αν η `isNew` επιστρέψει τιμή `true`, τότε είναι η πρώτη επίσκεψη του χρήστη (τουλάχιστον μέχρι τη λήξη της συνεδρίας). Αν όμως η μέθοδος `isNew` επιστρέψει τιμή `false`, αυτό απλώς δείχνει ότι έχει γίνει επίσκεψη στην εφαρμογή Ιστού, και όχι ότι ο χρήστης έχει επισκεφθεί τη μικροϋπηρεσία σας ή τη σελίδα JSP.

public long getCreationTime()

Αυτή η μέθοδος επιστρέφει τη χρονική στιγμή που δημιουργήθηκε για πρώτη φορά η συνεδρία, σε χιλιοστά του δευτερολέπτου από τα μεσάνυχτα της 1ης Ιανουαρίου του 1970 (GMT). Για να πάρετε μια τιμή που είναι κατάλληλη για εκτύπωση, μεταβιβάστε αυτή την τιμή στη συνάρτηση δόμησης `Date` ή στη μέθοδο `setTimeInMillis` της κλάσης `GregorianCalendar`.

public long getLastAccessedTime()

Αυτή η μέθοδος επιστρέφει τη χρονική στιγμή στην οποία προσπελάστηκε για τελευταία φορά η συνεδρία από τον πελάτη, σε χιλιοστά του δευτερολέπτου από τα μεσάνυχτα της 1ης Ιανουαρίου του 1970.

public int getMaxInactiveInterval()
public void setMaxInactiveInterval(int seconds)

Αυτές οι μέθοδοι λαμβάνουν ή ορίζουν τη χρονική διάρκεια, σε δευτερόλεπτα, κατά την οποία η συνεδρία θα μπορεί να παραμείνει χωρίς προσπέλαση πριν ακυρωθεί αυτόματα. Η αρνητική τιμή καθορίζει ότι η συνεδρία δεν θα πρέπει να λήξει ποτέ. Σημειώ-

στε ότι ο χρόνος παραμονής διατηρείται στο διακομιστή, και *δεν* είναι ίδιος με την ημερομηνία λήξης του μπισκότου. Ο ένας λόγος είναι ότι οι συνεδρίες κανονικά βασίζονται σε μπισκότα στη μνήμη, και όχι σε διατηρούμενα μπισκότα, οπότε *δεν* υπάρχει ημερομηνία λήξης. Ακόμα και αν παρεμβαίνατε στο μπισκότο `JSESSIONID` και το στέλνατε με κάποια ημερομηνία λήξης, οι συνεδρίες φυλλομετρητή και οι συνεδρίες διακομιστή είναι δύο τελείως διαφορετικά πράγματα. Για λεπτομέρειες όσον αφορά αυτή τη διάκριση, δείτε την επόμενη ενότητα.

9.4 Σύγκριση συνεδριών φυλλομετρητή και συνεδριών διακομιστή

Εξ ορισμού, η παρακολούθηση συνεδριών βασίζεται σε μπισκότα που βρίσκονται στη μνήμη του φυλλομετρητή, και όχι στο δίσκο. Έτσι, εκτός και αν η μικροϋπηρεσία διαβάσει ρητά το εισερχόμενο μπισκότο `JSESSIONID`, ορίσει τη μέγιστη διάρκεια ζωής και τη διαδρομή, και το στείλει πίσω, ο τερματισμός του φυλλομετρητή έχει αποτέλεσμα τη διακοπή της συνεδρίας: ο πελάτης δεν θα μπορεί πλέον να προσπελάσει τη συνεδρία. Το πρόβλημα, όμως, είναι ότι ο διακομιστής δεν γνωρίζει ότι ο φυλλομετρητής έχει κλείσει, και έτσι ο διακομιστής θα πρέπει να διατηρήσει στη μνήμη του τη συνεδρία μέχρι να γίνει υπέρβαση του χρονικού ορίου αδράνειας.

Φανταστείτε μια κανονική επίσκεψη σε ένα κατάστημα. Κοιτάτε γύρω σας, τοποθετείτε τα είδη σε ένα φυσικό καρότσι αγορών, και μετά αφήνετε το καρότσι στην άκρη ενός διαδρόμου για να δείτε κάποιο άλλο είδος. Ένας υπάλληλος πλησιάζει και βλέπει το καρότσι. Μπορεί να βάλει πίσω στα ράφια τα είδη; Όχι, επειδή συνεχίζετε να ψωνίζετε και θα επιστρέψετε σύντομα στο καρότσι σας. Τι θα συμβεί στην περίπτωση που συνειδητοποιείτε ότι έχετε ξεχάσει το πορτοφόλι σας, οπότε μπαίνετε στο αυτοκίνητό σας και επιστρέφετε στο σπίτι σας; Μπορεί τότε ο υπάλληλος να τοποθετήσει τα είδη του καροτσιού σας πίσω στα ράφια; Και πάλι όχι, επειδή ο υπάλληλος μάλλον *δεν γνωρίζει* ότι έχετε φύγει από το κατάστημα. Τι θα κάνει τότε ο υπάλληλος; Μπορεί να παρακολουθεί το καρότσι και, αν κανείς δεν το αγγίξει για κάποια χρονική περίοδο, μπορεί να συμπεράνει ότι το καρότσι έχει εγκαταλειφθεί, οπότε μπορεί να βγάλει τα πράγματα από αυτό. Η μόνη εξαίρεση είναι αν πάτε το καρότσι σας στον υπάλληλο και του πείτε "Λυπάμαι, αλλά ξέχασα το πορτοφόλι μου στο σπίτι, γι' αυτό και θα πρέπει να φύγω."

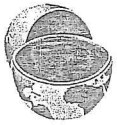
Η ανάλογη κατάσταση στον κόσμο των μικροϋπηρεσιών είναι όταν ο διακομιστής προσπαθεί να αποφασίσει αν μπορεί να απορρίψει το αντικείμενό σας `HttpSession`. Το ότι τη δεδομένη στιγμή δεν χρησιμοποιείτε τη συνεδρία δεν σημαίνει ότι ο διακομιστής μπορεί να την απορρίψει. Ίσως επιστρέψετε σύντομα (υποβάλετε μια νέα αίτηση). Αν κλείσετε το φυλλομετρητή σας, οπότε τα μπισκότα συνεδρίας σε επίπεδο φυλλομετρητή καταστρέφονται, ουσιαστικά η συνεδρία έχει διακοπεί. Όμως, όπως και στην περίπτωση του υπαλλήλου του καταστήματος που αναφέραμε προηγουμένως, ο διακομιστής *δεν γνωρίζει* ότι έχετε κλείσει το φυλλομετρητή. Έτσι ο διακομιστής θα πρέπει να περιμένει για κάποιο χρονικό διάστημα, για να δει αν η συνεδρία έχει εγκαταλειφθεί. Οι συνεδρίες ακυρώνονται αυτόματα όταν ο χρόνος μεταξύ των προσπελάσεων του πελάτη υπερβεί την τιμή που έχει καθοριστεί από τη μέθοδο `getMaxInactiveInterval`. Όταν συμβεί αυτό, τα αντικείμενα που είναι αποθηκευμένα στο αντικείμενο `HttpSession` αποδεσμεύονται. Στη συνέχεια, αν αυτά τα αντικείμενα υλοποιούν τη διασύνδεση `HttpSessionBindingListener`, ειδοποιούνται αυτόματα. Η μοναδική εξαίρεση στον κανόνα "ο διακομιστής περιμένει μέχρι να λήξουν οι συνεδρίες" είναι αν κληθούν οι μέθοδοι `invalidate` ή `logout`. Αυτό μοιάζει με την περίπτωση που λέτε στον υπάλληλο του καταστήματος ότι φεύγετε, οπότε ο διακομιστής αφαιρεί αμέσως όλα τα στοιχεία από τη συνεδρία και καταστρέφει το αντικείμενο της συνεδρίας.

9.5 Κωδικοποίηση URL που στάλθηκαν στο χρήστη

Εξ ορισμού, οι αποδέκτες μικροϋπηρεσιών (μηχανές) χρησιμοποιούν ως μηχανισμό για την παρακολούθηση συνεδριών τα μπισκότα. Υποθέστε, όμως, ότι αλλάζετε τη διευθέτηση του διακομιστή σας έτσι ώστε να χρησιμοποιεί την επανεγγραφή URL. Ποιες θα είναι οι αλλαγές στον κώδικά σας;

Τα καλά νέα: ο πυρήνας του κώδικα παρακολούθησης συνεδριών δεν χρειάζεται να αλλάξει καθόλου.

Τα κακά νέα: ένα μεγάλο μέρος του *υπόλοιπου* κώδικα θα πρέπει να αλλάξει. Συγκεκριμένα, αν κάποιες από τις σελίδες σας περιέχουν συνδέσμους προς την τοποθεσία σας, θα πρέπει να προσθέσετε ρητά τα δεδομένα συνεδριών στις διευθύνσεις URL. Βέβαια, η API των μικροϋπηρεσιών παρέχει μεθόδους για την προσθήκη αυτών των πληροφοριών σε οποιοδήποτε URL που θα καθορίσετε. Το πρόβλημα είναι ότι θα πρέπει να *καλέσετε* αυτές τις μεθόδους· από τεχνικής πλευράς δεν είναι εφικτό για το σύστημα να εξετάσει την έξοδο όλων των μικροϋπηρεσιών σας και των σελίδων JSP, να βρει ποια τμήματα περιέχουν υπερσυνδέσμους για την τοποθεσία σας, και να τροποποιήσει αυτά τα URL. Θα πρέπει να πείτε εσείς στο σύστημα ποια URL θα πρέπει να τροποποιήσει. Αυτή η απαίτηση σημαίνει ότι δεν μπορείτε να έχετε στατικές σελίδες HTML όταν χρησιμοποιείτε την επανεγγραφή URL για την παρακολούθηση συνεδριών, ή τουλάχιστον δεν μπορείτε να έχετε στατικές σελίδες HTML που κάνουν αναφορά στην τοποθεσία σας. Για πολλές εφαρμογές κάτι τέτοιο αποτελεί σημαντική επιβάρυνση, αλλά σε μερικές περιπτώσεις αξίζει τον κόπο.



Προειδοποίηση

Αν χρησιμοποιείτε την επανεγγραφή διευθύνσεων URL για την παρακολούθηση συνεδριών, οι περισσότερες ή όλες οι σελίδες σας θα πρέπει να παράγονται δυναμικά. Δεν μπορείτε να έχετε στατικές σελίδες HTML που περιέχουν υπερσυνδέσμους σε δυναμικές σελίδες της τοποθεσίας σας.

Υπάρχουν δύο πιθανές περιπτώσεις στις οποίες μπορεί να χρησιμοποιήσετε URL που αναφέρονται στην τοποθεσία σας.

Η πρώτη περίπτωση είναι όταν τα URL είναι ενσωματωμένα στην ιστοσελίδα που παράγει η μικροϋπηρεσία. Αυτά τα URL θα πρέπει να μεταβιβαστούν στη μέθοδο `encodeURL` της κλάσης `HttpServletResponse`. Η μέθοδος προσδιορίζει αν χρησιμοποιείται επανεγγραφή URL και

προσαρτά τις πληροφορίες της συνεδρίας μόνο όταν είναι απαραίτητο. Διαφορετικά το URL επιστρέφεται αμετάβλητο.

Ας δούμε ένα παράδειγμα:

```
String originalURL = someRelativeOrAbsoluteURL;
String encodeURL = response.encodeURL(originalURL);
out.println("<A HREF="\" + encodeURL + "\">...</A>");
```

Η δεύτερη περίπτωση στην οποία μπορεί να χρησιμοποιήσετε ένα URL που αναφέρεται στην τοποθεσία σας είναι η κλήση `sendRedirect` (δηλαδή, να έχει τοποθετηθεί το URL στην κεφαλίδα απάντησης `Location`). Σε αυτή τη δεύτερη περίπτωση υπάρχουν διαφορετικοί κανόνες που προσδιορίζουν αν οι πληροφορίες συνεδρίας θα πρέπει να προσαρτηθούν, οπότε δεν μπορείτε να χρησιμοποιήσετε τη μέθοδο `encodeURL`. Ευτυχώς, η κλάση `HttpServletResponse` παρέχει τη μέθοδο `encodeRedirectURL` για το χειρισμό αυτής της περίπτωσης. Ας δούμε ένα παράδειγμα:

```
String originalURL = someURL;
String encodeURL = response.encodeRedirectURL(originalURL);
response.sendRedirect(encodeURL);
```

Αν νομίζετε ότι υπάρχει η πιθανότητα η εφαρμογή Ιστού να χρησιμοποιήσει τελικά την επανεγγραφή URL αντί των μπισκότων, είναι καλή πρακτική να προνοήσετε και να κωδικοποιήσετε τα URL που αναφέρονται στην τοποθεσία σας.

9.6 Μια μικροϋπηρεσία που παρουσιάζει τις επισκέψεις ανά πελάτη

Η Λίστα 9.1 παρουσιάζει μια απλή μικροϋπηρεσία που δείχνει τις βασικές πληροφορίες σχετικά με τη συνεδρία ενός πελάτη. Όταν συνδέεται ο πελάτης, η μικροϋπηρεσία χρησιμοποιεί τη μέθοδο `request.getSession` είτε για να ανακτήσει την υπάρχουσα συνεδρία, είτε για να δημιουργήσει μια καινούργια συνεδρία, αν δεν υπάρχει παλιά. Κατόπιν η μικροϋπηρεσία αναζητεί την ιδιότητα `accessCount`, τύπου `Integer`. Αν δεν υπάρχει αυτή η ιδιότητα χρησιμοποιεί το 0 ως πλήθος προηγούμενων επισκέψεων. Αυτή η τιμή αυξάνεται στη συνέχεια και συσχετίζεται με τη συνεδρία μέσω της μεθόδου `setAttribute`. Τέλος η μικροϋπηρεσία τυπώνει ένα μικρό πίνακα HTML στον οποίο φαίνονται πληροφορίες σχετικά με τη συνεδρία.

Σημειώστε ότι ο τύπος `Integer` είναι μια *αμετάβλητη* (immutable) δομή δεδομένων: από τη στιγμή που θα κατασκευαστεί, δεν μπορεί να τροποποιηθεί. Αυτό σημαίνει ότι πρέπει να εκχωρήσετε ένα νέο αντικείμενο `Integer` για κάθε αίτηση, και μετά να χρησιμοποιήσετε τη μέθοδο `setAttribute` για να αντικαταστήσετε το παλιό αντικείμενο. Το απόσπασμα κώδικα που ακολουθεί παρουσιάζει τη γενική προσέγγιση παρακολούθησης συνεδρίας όταν αποθηκεύεται ένα αμετάβλητο αντικείμενο.

```
HttpSession session = request.getSession();
SomeImmutableClass value =
  (SomeImmutableClass)session.getAttribute("someIdentifier");
if (value == null) { //Δεν υπάρχει αυτό το αντικείμενο στη συνεδρία
  value = new SomeImmutableClass(...);
} else {
  value = new SomeImmutableClass(calculatedFrom(value));
}
session.setAttribute("someIdentifier", value);
doSomethingWith(value);
```


Αυτή η προσέγγιση βρίσκεται σε αντίθεση με την προσέγγιση που χρησιμοποιείται στην επόμενη ενότητα (Ενότητα 9.7) όπου έχουμε μεταβλητή (mutable) δομή δεδομένων. Σε αυτή την προσέγγιση εκχωρείται το αντικείμενο και καλείται η `setAttribute` μόνο όταν δεν υπάρχει τέτοιο αντικείμενο στη συνεδρία. Με άλλα λόγια, τα *περιεχόμενα* του αντικειμένου αλλάζουν κάθε φορά, αλλά η συνεδρία διατηρεί την ίδια *αναφορά* αντικειμένου.

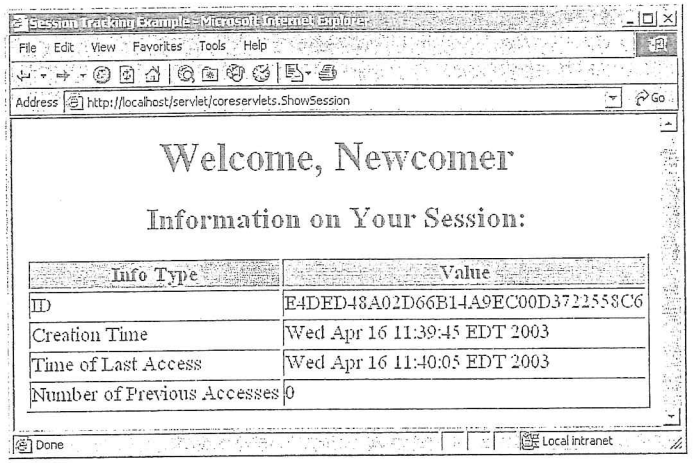
Οι Εικόνες 9-1 και 9-2 δείχνουν τη μικροϋπηρεσία κατά την αρχική επίσκεψη καθώς και στη συνέχεια, αφού ξαναφορτώσουμε τη σελίδα μερικές φορές.

Λίστα 9.1 ShowSession.java

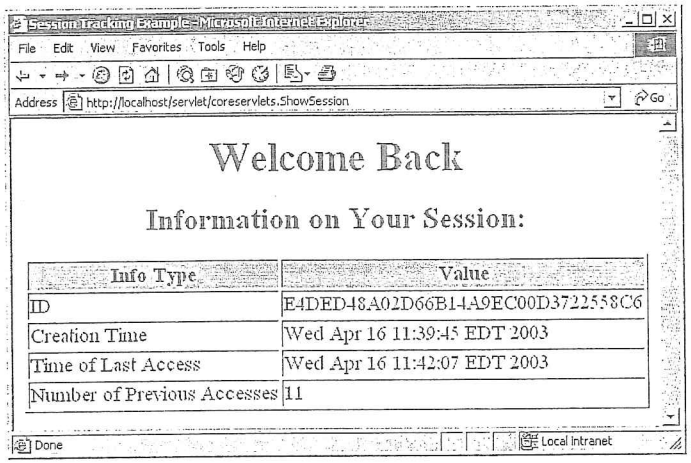
```
package coreservlets;

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

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

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

Λίστα 9.1 ShowSession.java *(συνέχεια)*

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

Εικόνα 9-1 Πρώτη επίσκεψη του πελάτη στη μικροϋπηρεσία ShowSession.



Εικόνα 9-2 Δωδέκατη επίσκεψη στη μικροϋπηρεσία ShowSession. Ο αριθμός των επισκέψεων αυτού του πελάτη είναι ανεξάρτητος από τις επισκέψεις άλλων πελατών.

9.7 Συγκέντρωση ενός καταλόγου με δεδομένα χρηστών

Στο παράδειγμα της προηγούμενης ενότητας (Ενότητα 9.6) αποθηκεύαμε τα δεδομένα του χρήστη στο αντικείμενο HttpSession του χρήστη. Το αποθηκευμένο αντικείμενο (τύπου Integer) είναι μια αμετάβλητη (immutable) δομή δεδομένων: μια δομή που δεν μπορεί να τρο-

ποποιηθεί. Έτσι για κάθε αίτηση εκχωρείται ένα νέο αντικείμενο `Integer`, και αυτό το νέο αντικείμενο αποθηκεύεται στη συνεδρία με τη μέθοδο `setAttribute` αντικαθιστώντας το παλαιό.

Μια άλλη συνηθισμένη προσέγγιση είναι η χρήση μιας *μεταβλητής* (mutable) δομής δεδομένων, όπως ο πίνακας, η λίστα (`List`), ο χάρτης (`Map`), ή κάποια ειδική για την εφαρμογή δομή δεδομένων που έχει εγγράψιμα πεδία (στιγμιότυπα μεταβλητών). Με αυτή την προσέγγιση δεν χρειάζεται να καλέσετε τη `setAttribute`, παρά μόνο όταν γίνεται εκχώρηση του αντικειμένου για πρώτη φορά. Το βασικό πρότυπο είναι το εξής:

```
HttpSession session = request.getSession();
SomeMutableClass value =
  (SomeMutableClass)session.getAttribute("someIdentifier");
if (value == null) { //Δεν υπάρχει το αντικείμενο στη συνεδρία
  value = new SomeMutableClass(...);
  session.setAttribute("someIdentifier", value);
}
value.updateInternalState(...);
doSomethingWith(value);
```

Οι μεταβλητές δομές δεδομένων χρησιμοποιούνται κυρίως για τη διατήρηση ενός συνόλου δεδομένων που σχετίζονται με το χρήστη. Σε αυτή την ενότητα θα παρουσιάσουμε ένα απλοποιημένο παράδειγμα στο οποίο θα διατηρούμε μια λίστα με είδη που έχει αγοράσει ο κάθε χρήστης. Στην επόμενη ενότητα (Ενότητα 9.8) θα παρουσιάζουμε ένα πλήρες παράδειγμα καλαθιού αγορών. Το μεγαλύτερο τμήμα του κώδικα σε αυτό το παράδειγμα αφορά την αυτόματη δόμηση των ιστοσελίδων που εμφανίζουν τα είδη και το ίδιο το καλάθι αγορών. Αν και αυτά τα ειδικά κομμάτια της εφαρμογής μπορεί να είναι κάπως περίπλοκα, ο κώδικας της βασικής παρακολούθησης συνεδρίας είναι αρκετά απλός. Ακόμα και έτσι, όμως, είναι χρήσιμο να δείτε τη θεμελιώδη προσέγγιση χωρίς να διασπάται η προσοχή σας από τα ειδικά κομμάτια της εφαρμογής. Αυτός είναι ο σκοπός του παραδείγματος αυτής της ενότητας.

Η Λίστα 9.2 παρουσιάζει μια εφαρμογή που χρησιμοποιεί ένα απλό αντικείμενο `ArrayList` (ο αντικαταστάτης του τύπου `Vector` στην πλατφόρμα Java 2) για την παρακολούθηση των ειδών τα οποία έχει αγοράσει ο κάθε χρήστης. Εκτός από την εύρεση ή τη δημιουργία μιας συνεδρίας και την εισαγωγή του νέου είδους προς αγορά σε αυτή (η τιμή της παραμέτρου αίτησης `newItem`), αυτό το παράδειγμα τυπώνει μια λίστα με κουκκίδες με τα είδη που βρίσκονται στο "καλάθι" (δηλαδή τα περιεχόμενα του `ArrayList`). Σημειώστε ότι ο κώδικας που τυπώνει τη λίστα είναι συγχρονισμένος με το `ArrayList`. Αξίζει τον κόπο να πάρετε αυτή την προφύλαξη, αν και θα πρέπει να έχετε υπόψη σας ότι οι περιπτώσεις στις οποίες είναι απαραίτητος ο συγχρονισμός είναι εξαιρετικά σπάνιες. Επειδή ο κάθε χρήστης έχει ξεχωριστή συνεδρία, ο μόνος τρόπος για να δημιουργηθεί κατάσταση ανταγωνισμού είναι να υποβάλει ο ίδιος χρήστης δύο αγορές τη μία αμέσως μετά την άλλη. Αν και είναι απίθανο, κάτι τέτοιο *είναι* δυνατό, οπότε ο συγχρονισμός έχει την αξία του.

Λίστα 9.2 ShowItems.java

```
package coreservlets;

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

/** Μικροϋπηρεσία που εμφανίζει μία λίστα παραγγελιών. Συγκεντρώνει
 *  τις παραγγελίες σε ένα ArrayList χωρίς να προσπαθεί να ανιχνεύσει
 *  επαναλαμβανόμενα είδη. Τη χρησιμοποιούμε για να δείξουμε τα βασικά
 *  στοιχεία για την παρακολούθηση συνεδριών.
*/

public class ShowItems extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    ArrayList previousItems =
      (ArrayList)session.getAttribute("previousItems");
    if (previousItems == null) {
      previousItems = new ArrayList();
      session.setAttribute("previousItems", previousItems);
    }
    String newItem = request.getParameter("newItem");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Items Purchased";
    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>" + title + "</H1>");
    synchronized(previousItems) {
      if ((newItem != null) &&
          (!newItem.trim().equals(""))) {
        previousItems.add(newItem);
      }
      if (previousItems.size() == 0) {
        out.println("<I>No items</I>");
      } else {
        out.println("<UL>");
        for(int i=0; i<previousItems.size(); i++) {
          out.println("<LI>" + (String)previousItems.get(i));
        }
```

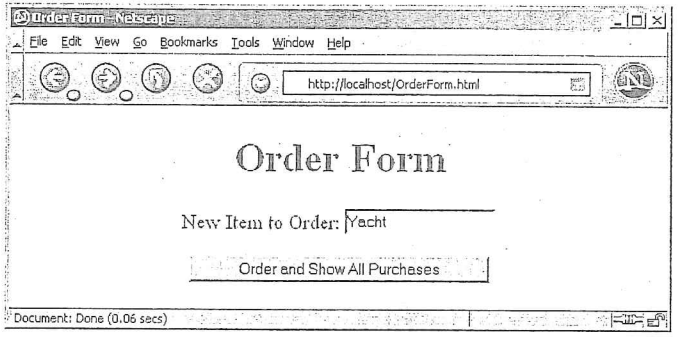
Λίστα 9.2 ShowItems.java *(συνέχεια)*

```
        out.println("</UL>");
      }
    }
    out.println("</BODY></HTML>");
  }
}
```

Η Λίστα 9.3 παρουσιάζει μια φόρμα HTML που συγκεντρώνει τιμές από την παράμετρο newItem και τις υποβάλει στη μικροϋπηρεσία. Η Εικόνα 9-3 δείχνει το αποτέλεσμα της φόρμας· οι Εικόνες 9-4 και 9-5 δείχνουν τα αποτελέσματα της μικροϋπηρεσίας πριν γίνει επίσκεψη στη φόρμα παραγγελιών και αφού γίνουν πολλές επισκέψεις, αντίστοιχα.

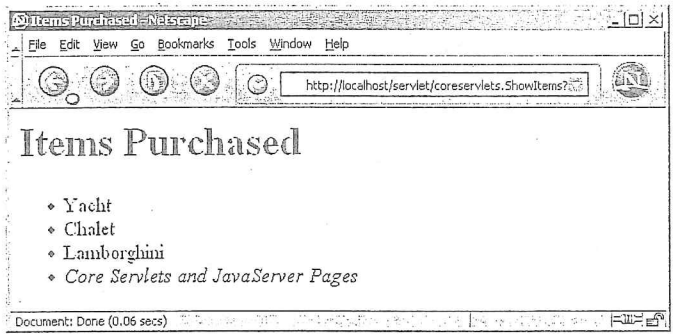
Λίστα 9.3 OrderForm.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Order Form</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1>Order Form</H1>
<FORM ACTION="servlet/coreservlets.ShowItems">
  New Item to Order:
  <INPUT TYPE="TEXT" NAME="newItem" VALUE="Yacht"><P>
  <INPUT TYPE="SUBMIT" VALUE="Order and Show All Purchases">
</FORM>
  </CENTER></BODY></HTML>
```



Εικόνα 9-3 Εμπροσθοφυλακή για τη μικροϋπηρεσία εμφάνισης ειδών.

Εικόνα 9-4 Η μικροϋπηρεσία εμφάνισης ειδών πριν γίνουν αγορές.



Εικόνα 9-5 Η μικροϋπηρεσία εμφάνισης ειδών αφού αγοραστούν μερικά μάλλον ακριβά είδη.

9.8 Ηλεκτρονικό κατάστημα με καλάθι αγορών και παρακολούθηση συνεδρίας

Σε αυτή την ενότητα θα παρουσιάσουμε ένα εκτενές παράδειγμα το οποίο δείχνει πώς θα μπορούσατε να δημιουργήσετε ένα ηλεκτρονικό κατάστημα που χρησιμοποιεί παρακολούθηση συνεδρίας. Η πρώτη υποενότητα παρουσιάζει τη δόμηση των σελίδων που εμφανίζουν τα είδη τα οποία διατίθενται προς πώληση. Ο κώδικας για κάθε εμφανιζόμενη σελίδα περιλαμβάνει απλώς τον τίτλο της σελίδας και τα αναγνωριστικά των ειδών που εμφανίζονται στη σελίδα. Η πραγματική σελίδα δομείται αυτόματα από μεθόδους της γονικής κλάσης, με βάση τις περιγραφές των ειδών που είναι αποθηκευμένες στον κατάλογο. Η δεύτερη υποενότητα παρουσιάζει ένα καλάθι αγορών για κάθε χρήστη και επιτρέπει στο χρήστη να τροποποιήσει τις παραγγελίες για τα ήδη επιλεγμένα είδη. Η τρίτη υποενότητα παρουσιάζει την υλοποίηση του καλαθιού αγορών, δηλαδή τις δομές δεδομένων που αντιπροσωπεύουν τα μεμονωμένα είδη και τις παραγγελίες, καθώς και τον κατάλογο.

Δημιουργία της εμπροσθοφυλακής

Η Λίστα 9.4 παρουσιάζει μια αφηρημένη βασική κλάση (abstract base class) που χρησιμοποιείται ως σημείο εκκίνησης για τις μικροϋπηρεσίες που θέλουν να εμφανίσουν είδη προς πώληση. Παίρνει τα αναγνωριστικά των ειδών προς πώληση, τα αναζητεί στον κατάλογο, και χρησιμοποιεί τις τιμές και τις περιγραφές που υπάρχουν εκεί για να παρουσιάσει στο χρήστη μια σελίδα παραγγελιών. Η Λίστα 9.5 (το αποτέλεσμα φαίνεται στην Εικόνα 9-6) και η Λίστα 9.6 (το αποτέλεσμα φαίνεται στην Εικόνα 9-7) δείχνουν πόσο εύκολη είναι η δόμηση πραγματικών σελίδων με αυτή τη γονική κλάση.

Λίστα 9.4 CatalogPage.java

```
package coreservlets;

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

/** Βασική κλάση για σελίδες που παρουσιάζουν καταχωρίσεις καταλόγου.
 *  Οι μικροϋπηρεσίες που επεκτείνουν αυτή την κλάση πρέπει να καθορίζουν
 *  τις καταχωρίσεις καταλόγου που πωλούν και τον τίτλο της σελίδας
 *  <I>πριν</I> γίνει προσπέλαση της μικροϋπηρεσίας. Αυτό γίνεται
 *  με την τοποθέτηση των κλήσεων setItems και setTitle
 *  στην init.
*/

public abstract class CatalogPage extends HttpServlet {
  private CatalogItem[] items;
  private String[] itemIDs;
  private String title;

  /** Με δεδομένο έναν πίνακα με αναγνωριστικά (ID) ειδών, αναζήτησή
   *  τους στο Catalog και τοποθέτηση των αντίστοιχων καταχωρίσεων
   *  CatalogItem στον πίνακα των ειδών. Ο CatalogItem περιέχει
   *  μια σύντομη περιγραφή, μια εκτενή περιγραφή, και μια τιμή,
   *  χρησιμοποιώντας το αναγνωριστικό του είδους ως μοναδικό κλειδί.
   *  <P>
   *  Οι μικροϋπηρεσίες που επεκτείνουν την CatalogPage θα <B>πρέπει</B>
   *  να καλέσουν αυτή τη μέθοδο (συνήθως από την init) πριν γίνει
   *  προσπέλαση της μικροϋπηρεσίας.
   */

  protected void setItems(String[] itemIDs) {
    this.itemIDs = itemIDs;
    items = new CatalogItem[itemIDs.length];
    for(int i=0; i<items.length; i++) {
      items[i] = Catalog.getItem(itemIDs[i]);
    }
  }
```


Λίστα 9.4 CatalogPage.java *(συνέχεια)*

```
  /** Ορίζει τον τίτλο της σελίδας, ο οποίος εμφανίζεται
   *  σε μία κεφαλίδα H1 της σελίδας που προκύπτει.
   *  <P>
   *  Οι μικροϋπηρεσίες που επεκτείνουν την CatalogPage θα <B>πρέπει</B>
   *  να καλέσουν αυτή τη μέθοδο (συνήθως από την init) πριν γίνει
   *  προσπέλαση της μικροϋπηρεσίας.
   */

  protected void setTitle(String title) {
    this.title = title;
  }

  /** Πρώτα εμφανίζεται ο τίτλος και μετά, για κάθε είδος του καταλόγου,
   *  τοποθετείται η σύντομη περιγραφή του σε μια επικεφαλίδα H2
   *  με την τιμή σε παρενθέσεις και την εκτενή περιγραφή από κάτω.
   *  Κάτω από κάθε καταχώριση τοποθετείται ένα κουμπί παραγγελίας,
   *  το οποίο υποβάλει στη μικροϋπηρεσία OrderPage τις πληροφορίες για
   *  την αντίστοιχη καταχώριση του καταλόγου.
   *  <P>
   *  Για να δείτε τον κώδικα HTML που προκύπτει από αυτή τη μέθοδο,
   *  επιλέξτε "View Source" (Προβολή πηγαίου κώδικα) στη σελίδα
   *  KidsBooksPage ή τη σελίδα TechBooksPage – δύο "συγκεκριμένες"
   *  κλάσεις που επεκτείνουν αυτή την αφηρημένη κλάση.
   */

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    if (items == null) {
      response.sendError(response.SC_NOT_FOUND,
                         "Missing Items.");
      return;
    }
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
    CatalogItem item;
    for(int i=0; i<items.length; i++) {
      out.println("<HR>");
      item = items[i];
```

Λίστα 9.4 CatalogPage.java *(συνέχεια)*

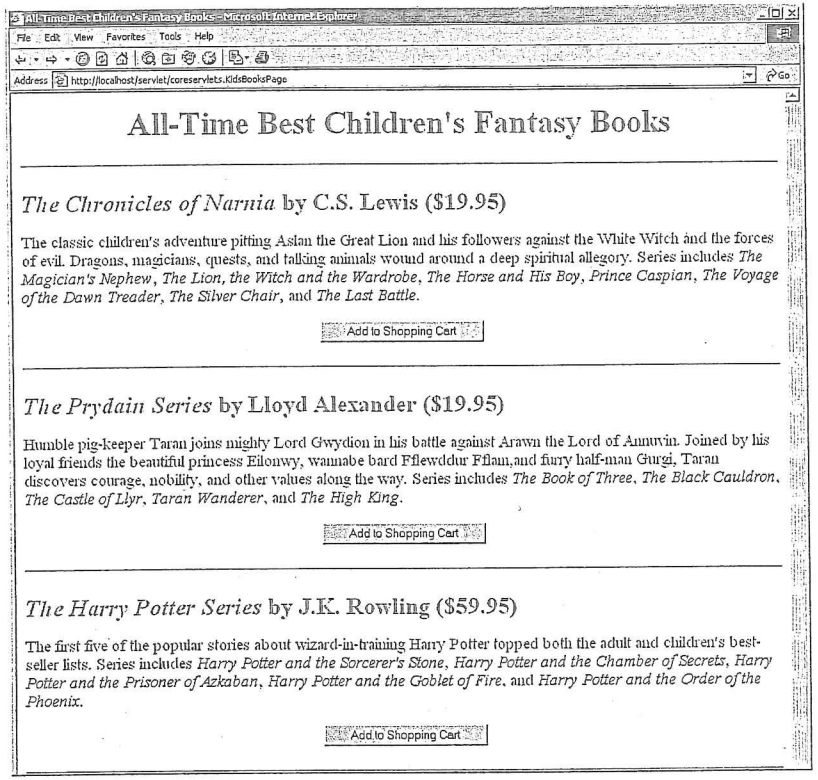
```
      // Εμφάνιση μηνύματος σφάλματος αν η υποκλάση περιέχει αναγνωριστικό
      // είδους που δεν υπάρχει στον κατάλογο.
      if (item == null) {
        out.println("<FONT COLOR=\"RED\">" +
                    "Unknown item ID " + itemIDs[i] +
                    "</FONT>");
      } else {
        out.println();
        String formURL =
          "/servlet/coreservlets.OrderPage";
        // Μεταβίβαση των URL που αναφέρονται στην τοποθεσία Ιστού
        // μέσω της encodeURL.
        formURL = response.encodeURL(formURL);
        out.println
          ("<FORM ACTION=\"" + formURL + "\">\n" +
           "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\" " +
           "       VALUE=\"" + item.getItemID() + "\">\n" +
           "<H2>" + item.getShortDescription() +
           " ($" + item.getCost() + ")</H2>\n" +
           item.getLongDescription() + "\n" +
           "<P>\n<CENTER>\n" +
           "<INPUT TYPE=\"SUBMIT\" " +
           "VALUE=\"Add to Shopping Cart\">\n" +
           "</CENTER>\n<P>\n</FORM>");
      }
    }
    out.println("<HR>\n</BODY></HTML>");
  }
}
```

Λίστα 9.5 KidsBooksPage.java

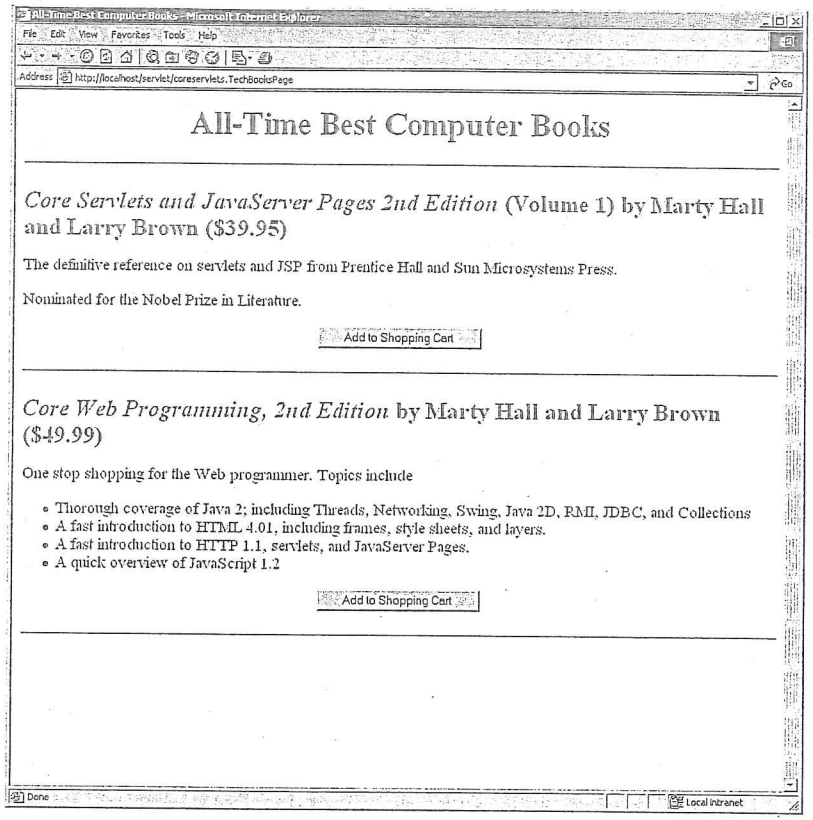
```
package coreservlets;

/** Εξειδίκευση της μικροϋπηρεσίας CatalogPage που εμφανίζει μια σελίδα
 *  για την πώληση τριών διάσημων σειρών παιδικών βιβλίων.
 *  Οι παραγγελίες στέλνονται στη μικροϋπηρεσία OrderPage.
*/

public class KidsBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "lewis001", "alexander001", "rowling001" };
    setItems(ids);
    setTitle("All-Time Best Children's Fantasy Books");
  }
}
```


Λίστα 9.6 TechBooksPage.java

```
package coreservlets;

/** Εξειδίκευση της μικροϋπηρεσίας CatalogPage που εμφανίζει μια σελίδα
 *  για την πώληση δύο διάσημων βιβλίων για υπολογιστές.
 *  Οι παραγγελίες στέλνονται στη μικροϋπηρεσία OrderPage.
*/

public class TechBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "hall001", "hall002" };
    setItems(ids);
    setTitle("All-Time Best Computer Books");
  }
}
```



Εικόνα 9-6 Αποτέλεσμα της μικροϋπηρεσίας KidsBooksPage.



Εικόνα 9-7 Αποτέλεσμα της μικροϋπηρεσίας TechBooksPage.

Χειρισμός παραγγελιών

Η Λίστα 9.7 παρουσιάζει τη μικροϋπηρεσία η οποία χειρίζεται τις παραγγελίες που προέρχονται από τις διάφορες σελίδες του καταλόγου, τον οποίο παρουσιάσαμε στην προηγούμενη υποενότητα. Το παράδειγμα αυτό χρησιμοποιεί την παρακολούθηση συνεδρίας για να συσχετίσει ένα καλάθι αγορών με κάθε χρήστη. Επειδή ο κάθε χρήστης έχει ξεχωριστή συνεδρία, είναι απίθανο πολλαπλά νήματα να προσπελάσουν ταυτόχρονα το ίδιο καλάθι αγορών. Ωστόσο θα μπορούσατε να φανταστείτε μερικές ακραίες περιπτώσεις κατά τις οποίες θα μπορούσε να συμβεί ταυτόχρονη προσπέλαση, όπως στην περίπτωση ενός χρήστη που έχει ανοιχτά πολλά παράθυρα του φυλλομετρητή και στέλνει διαδοχικές ενημερώσεις σε περισσότερα από ένα παράθυρα. Έτσι, για λόγους ασφαλείας, ο κώδικας συγχρονίζει τις προσπελάσεις με βάση το αντικείμενο συνεδρίας. Αυτός ο συγχρονισμός παρεμποδίζει άλλα νήματα που χρησιμοποιούν την ίδια συνεδρία να προσπελάσουν ταυτόχρονα τα δεδομένα, ενώ παράλληλα επιτρέπει να προχωρήσουν οι ταυτόχρονες

αιτήσεις από διαφορετικούς χρήστες. Οι Εικόνες 9-8 και 9-9 δείχνουν μερικά τυπικά αποτελέσματα.

Λίστα 9.7 OrderPage.java

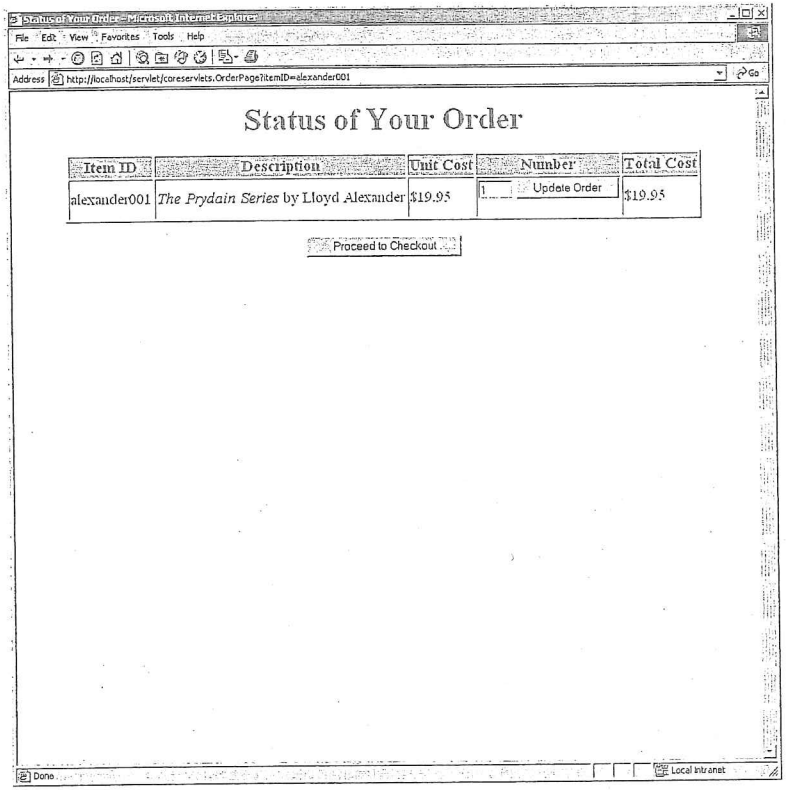
```
package coreservlets;

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

/** Εμφανίζει όλα τα είδη που βρίσκονται στο ShoppingCart. Οι πελάτες
 *  έχουν τη δική τους συνεδρία που παρακολουθεί ποιο ShoppingCart είναι
 *  δικό τους. Αν αυτή είναι η πρώτη επίσκεψη στη σελίδα παραγγελιών,
 *  δημιουργείται ένα καινούργιο καλάθι αγορών. Συνήθως οι πελάτες
 *  φτάνουν σε αυτή τη σελίδα μέσω μιας σελίδας που παρουσιάζει τις
 *  καταχωρίσεις του καταλόγου, οπότε αυτή η σελίδα προσθέτει ένα ακόμα
 *  είδος στο καλάθι αγορών. Όμως οι χρήστες μπορούν να δημιουργήσουν ένα
 *  σελιδοδείκτη για αυτή τη σελίδα, να την προσπελάσουν από τη λίστα του
 *  ιστορικού, ή να μεταφερθούν σε αυτή πατώντας σε κάποιο κουμπί
 *  "Update Order" (ενημέρωση παραγγελίας) μετά από αλλαγή του αριθμού
*   των ειδών τα οποία παραγγέλνουν.
*/

public class OrderPage extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession();
    ShoppingCart cart;
    synchronized(session) {
      cart = (ShoppingCart)session.getAttribute("shoppingCart");
      // Οι νέοι επισκέπτες λαμβάνουν ένα καινούργιο καλάθι αγορών.
      // Οι παλαιοί επισκέπτες συνεχίζουν να χρησιμοποιούν το καλάθι
τους.
      if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("shoppingCart", cart);
      }
      String itemID = request.getParameter("itemID");
      if (itemID != null) {
        String numItemsString =
          request.getParameter("numItems");
        if (numItemsString == null) {
          // Αν η αίτηση καθόρισε ένα αναγνωριστικό αλλά όχι αριθμό,
          // τότε οι πελάτες έρχονται εδώ μέσω ενός κουμπιού "Add Item to
          // Cart" στη σελίδα του καταλόγου.
          cart.addItem(itemID);
        } else {
```

Λίστα 9.7 OrderPage.java *(συνέχεια)*

```
      // Αν η αίτηση καθόρισε αναγνωριστικό και αριθμό, τότε οι
      // πελάτες έρχονται εδώ μέσω ενός κουμπιού "Update Order"
      // μετά από αλλαγή του αριθμού των ειδών στην παραγγελία.
      // Σημειώστε ότι ο αριθμός 0 έχει αποτέλεσμα
      // τη διαγραφή του είδους από το καλάθι.
      int numItems;
      try {
        numItems = Integer.parseInt(numItemsString);
      } catch(NumberFormatException nfe) {
        numItems = 1;
      }
      cart.setNumOrdered(itemID, numItems);
    }
  }
}
// Είτε άλλαξε είτε όχι την παραγγελία ο πελάτης,
// εμφάνιση της κατάστασης της παραγγελίας.
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Status of Your Order";
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>");
synchronized(session) {
  List itemsOrdered = cart.getItemsOrdered();
  if (itemsOrdered.size() == 0) {
    out.println("<H2><I>No items in your cart...</I></H2>");
  } else {
    // Αν υπάρχει τουλάχιστον ένα είδος στο καλάθι, εμφάνιση
    // του πίνακα με τα είδη τα οποία παραγγέλνει ο πελάτης.
    out.println
      ("<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
       "<TR BGCOLOR=\"#FFAD00\">\n" +
       "  <TH>Item ID<TH>Description\n" +
       "  <TH>Unit Cost<TH>Number<TH>Total Cost");
    ItemOrder order;
    // Στρογγυλοποίηση στα δύο δεκαδικά ψηφία, εισαγωγή του συμβόλου
    // του δολαρίου (ή άλλου νομίσματος), κ.λπ. ανάλογα με τις
    // τρέχουσες τοπικές ρυθμίσεις.
    NumberFormat formatter =
      NumberFormat.getCurrencyInstance();
    // Για κάθε καταχώριση στο καλάθι αγορών, δημιουργία μιας γραμμής
```


Λίστα 9.7 OrderPage.java *(συνέχεια)*

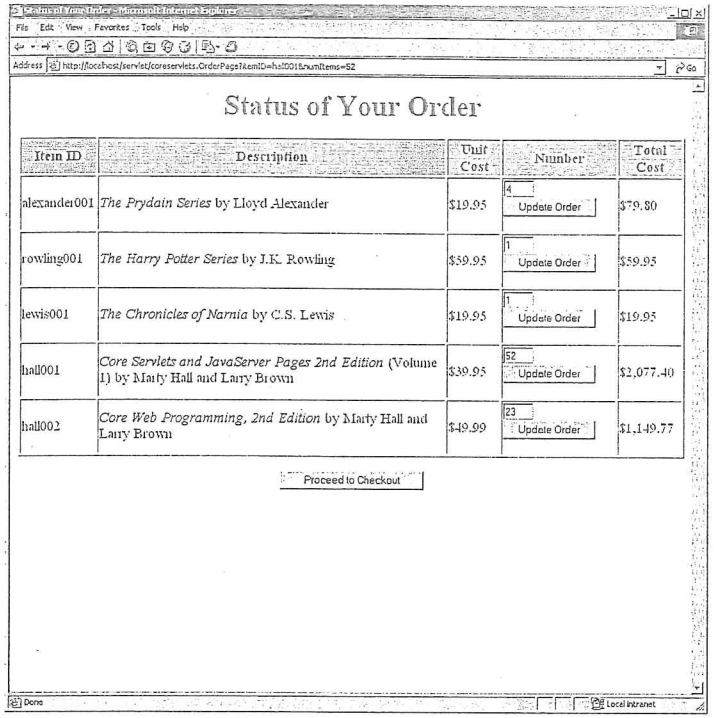
```
    // στον πίνακα με το αναγνωριστικό, την περιγραφή, το κόστος ανά
    // είδος, το πλήθος των ειδών, και το συνολικό κόστος.
    // Τοποθέτηση του πλήθους των ειδών σε ένα πεδίο κειμένου έτσι
    // ώστε ο χρήστης να μπορεί να το αλλάξει, με ένα κουμπί "Update
    // Order" δίπλα από αυτό, το οποίο υποβάλλει ξανά την ίδια
    // σελίδα αλλά καθορίζει διαφορετικό αριθμό αντικειμένων.
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.get(i);
      out.println
        ("<TR>\n" +
         "  <TD>" + order.getItemID() + "\n" +
         "  <TD>" + order.getShortDescription() + "\n" +
```



Εικόνα 9-8 Αποτέλεσμα της μικροϋπηρεσίας OrderPage όταν ο χρήστης πατά στο κουμπί "Add to Shopping Cart" της σελίδας KidsBooksPage.

Λίστα 9.7 OrderPage.java *(συνέχεια)*

```
            "  <TD>" +
            formatter.format(order.getUnitCost()) + "\n" +
            "  <TD>" +
            "<FORM>\n" +  // Submit to current URL
            "<INPUT TYPE=\"HIDDEN\" NAME=\"itemID\"\n" +
            "       VALUE=\"" + order.getItemID() + "\">\n" +
            "<INPUT TYPE=\"TEXT\" NAME=\"numItems\"\n" +
            "       SIZE=3 VALUE=\"" +
            order.getNumItems() + "\">\n" +
            "<SMALL>\n" +
            "<INPUT TYPE=\"SUBMIT\"\n "+
            "       VALUE=\"Update Order\">\n" +
            "</SMALL>\n" +
            "</FORM>\n" +
            "  <TD>" +
            formatter.format(order.getTotalCost()));
      }
      String checkoutURL =
        response.encodeURL("../Checkout.html");
      // Το κουμπί "Proceed to Checkout" κάτω από τον πίνακα
      out.println
        ("</TABLE>\n" +
         "<FORM ACTION=\"" + checkoutURL + "\">\n" +
         "<BIG><CENTER>\n" +
         "<INPUT TYPE=\"SUBMIT\"\n" +
         "       VALUE=\"Proceed to Checkout\">\n" +
         "</CENTER></BIG></FORM>");
    }
    out.println("</BODY></HTML>");
  }
}
 }
```

Εικόνα 9-9 Αποτέλεσμα της μικροϋπηρεσίας OrderPage μετά από προσθήκες και αλλαγές στην παραγγελία.

Λίστα 9.8 Checkout.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Checking Out</TITLE>
</HEAD>
<BODY BGCOLOR="#fdf5e6">
<H1 ALIGN="CENTER">Checking Out</h1>
We are sorry, but our electronic credit-card-processing
system is currently out of order. Please send a check
to:
```

Λίστα 9.8 Checkout.html *(συνέχεια)*

```
</PRE>
  Marty Hall
  coreservlets.com, Inc.
  6 Meadowsweet Ct., Suite B1
  Reisterstown, MD 21136-6020
  410-429-5535
  hall@coreservlets.com
</PRE>
Since we have not yet calculated shipping charges, please
sign the check but do not fill in the amount. We will
generously do that for you.
   </BODY></HTML>
```

Στα παρασκήνια: Υλοποίηση του καλαθιού αγορών και του καταλόγου ειδών

Η Λίστα 9.9 δείχνει την υλοποίηση του καλαθιού αγορών. Διατηρεί απλώς μια λίστα παραγγελιών (τύπου List), με μεθόδους για προσθήκη και ενημέρωση αυτών των παραγγελιών. Η Λίστα 9.10 παρουσιάζει τον κώδικα για ένα μεμονωμένο είδος του καταλόγου, η Λίστα 9.11 παρουσιάζει την κλάση που αντιπροσωπεύει την κατάσταση της παραγγελίας για ένα συγκεκριμένο είδος, και η Λίστα 9.12 δείχνει την υλοποίηση του καταλόγου.

Λίστα 9.9 ShoppingCart.java

```
package coreservlets;

import java.util.*;

/** Η δομή δεδομένων του καλαθιού αγορών που χρησιμοποιείται για την
 *  παρακολούθηση των παραγγελιών. Η μικροϋπηρεσία OrderPage συσχετίζει
 *  καθένα από αυτά τα καλάθια με κάθε συνεδρία χρήστη.
 */

public class ShoppingCart {
  private ArrayList itemsOrdered;

  /** Δόμηση κενού καλαθιού αγορών. */

  public ShoppingCart() {
    itemsOrdered = new ArrayList();
  }
```


Λίστα 9.9 ShoppingCart.java *(συνέχεια)*

```
/** Επιστρέφει μια λίστα (List) καταχωρίσεων ItemOrder με το
 *  είδος (Item) και το πλήθος που παραγγέλνεται. Δηλώνεται ως List
 *  και όχι ως ArrayList, έτσι ώστε να μπορεί αργότερα να αλλαχθεί
 *  ο υποκείμενος μηχανισμός.
 */

public List getItemsOrdered() {
  return(itemsOrdered);
}

/** Εξετάζει το καλάθι για να δει αν υπάρχει ήδη κάποια καταχώριση
 *  παραγγελίας που να αντιστοιχεί στο αναγνωριστικό του είδους. Αν
 *  υπάρχει, αυξάνει το πλήθος αντικειμένων που παραγγέλνονται. Αν όχι,
 *  αναζητεί το είδος (Item) στον κατάλογο και προσθέτει μια καταχώριση
 *  παραγγελίας για αυτό.
 */

public synchronized void addItem(String itemID) {
  ItemOrder order;
  for(int i=0; i<itemsOrdered.size(); i++) {
    order = (ItemOrder)itemsOrdered.get(i);
    if (order.getItemID().equals(itemID)) {
      order.incrementNumItems();
      return;
    }
  }
  ItemOrder newOrder = new ItemOrder(Catalog.getItem(itemID));
  itemsOrdered.add(newOrder);
}

/** Εξετάζει το καλάθι για να βρει καταχώριση παραγγελίας που να
 *  αντιστοιχεί στο αναγνωριστικό του είδους. Αν ο προσδιοριζόμενος
 *  αριθμός είναι θετικός, τον ορίζει. Αν ο αριθμός που δίνεται είναι 0
 *  (ή αρνητικός, λόγω εσφαλμένης εισόδου του χρήστη) διαγράφει
 *  το είδος από το καλάθι.
 */

public synchronized void setNumOrdered(String itemID,
                                       int numOrdered) {
  ItemOrder order;
  for(int i=0; i<itemsOrdered.size(); i++) {
    order = (ItemOrder)itemsOrdered.get(i);
    if (order.getItemID().equals(itemID)) {
```

Λίστα 9.9 ShoppingCart.java *(συνέχεια)*

```
        if (numOrdered <= 0) {
          itemsOrdered.remove(i);
        } else {
          order.setNumItems(numOrdered);
        }
        return;
      }
    }
    ItemOrder newOrder =
      new ItemOrder(Catalog.getItem(itemID));
    itemsOrdered.add(newOrder);
  }
}
```

Λίστα 9.10 CatalogItem.java

```
package coreservlets;

/** Περιγράφει ένα είδος του καταλόγου για το ηλεκτρονικό κατάστημα.
 *  Το itemID προσδιορίζει με μοναδικό τρόπο το είδος, η σύντομη
 *  περιγραφή δίνει σύντομες πληροφορίες (όπως ο τίτλος του βιβλίου
 *  και ο συγγραφέας), η εκτενής περιγραφή περιγράφει το είδος με λίγες
 *  προτάσεις,και το cost δίνει την τρέχουσα τιμή ανά είδος. Τόσο η
 *  σύντομη όσο και η εκτενής περιγραφή μπορούν να περιέχουν στοιχεία HTML.
 *  <P>
*/

public class CatalogItem {
  private String itemID;
  private String shortDescription;
  private String longDescription;
  private double cost;

  public CatalogItem(String itemID, String shortDescription,
                     String longDescription, double cost) {
    setItemID(itemID);
    setShortDescription(shortDescription);
    setLongDescription(longDescription);
    setCost(cost);
  }

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

  protected void setItemID(String itemID) {
    this.itemID = itemID;
  }
```


Λίστα 9.10 CatalogItem.java *(συνέχεια)*

```
  public String getShortDescription() {
    return(shortDescription);
  }

  protected void setShortDescription(String shortDescription) {
    this.shortDescription = shortDescription;
  }

  public String getLongDescription() {
    return(longDescription);
  }

  protected void setLongDescription(String longDescription) {
    this.longDescription = longDescription;
  }

  public double getCost() {
    return(cost);
  }

  protected void setCost(double cost) {
    this.cost = cost;
  }
}
```

Λίστα 9.11 ItemOrder.java

```
package coreservlets;

/** Συσχετίζει ένα είδος (Item) του καταλόγου με μια συγκεκριμένη
 *  παραγγελία παρακολουθώντας το πλήθος των παραγγελθέντων και το
 *  συνολικό κόστος. Παρέχει επίσης κάποιες βολικές μεθόδους για τη
 *  λήψη των δεδομένων από το CatalogItem χωρίς ξεχωριστή εξαγωγή
 *  του CatalogItem.
 */

public class ItemOrder {
  private CatalogItem item;
  private int numItems;

  public ItemOrder(CatalogItem item) {
    setItem(item);
    setNumItems(1);
  }
```

Λίστα 9.11 ItemOrder.java *(συνέχεια)*

```
public CatalogItem getItem() {
  return(item);
}

protected void setItem(CatalogItem item) {
  this.item = item;
}

public String getItemID() {
  return(getItem().getItemID());
}

public String getShortDescription() {
  return(getItem().getShortDescription());
}

public String getLongDescription() {
  return(getItem().getLongDescription());
}

public double getUnitCost() {
  return(getItem().getCost());
}

public int getNumItems() {
  return(numItems);
}

public void setNumItems(int n) {
  this.numItems = n;
}

public void incrementNumItems() {
  setNumItems(getNumItems() + 1);
}

public void cancelOrder() {
  setNumItems(0);
}

public double getTotalCost() {
  return(getNumItems() * getUnitCost());
}
}
```


Λίστα 9.12 Catalog.java

```
package coreservlets;

/** Ένας κατάλογος που παραθέτει τα διαθέσιμα είδη στην αποθήκη. */

public class Catalog {
  // Στην πραγματική ζωή, αυτό θα προέρχεται από κάποια βάση δεδομένων.
  // Χρησιμοποιούμε στατικό πίνακα για λόγους ευκολίας στον έλεγχο και
  // την εκδίπλωση της εφαρμογής. Για πληροφορίες σχετικά με τη χρήση
  // βάσεων δεδομένων σε μικροϋπηρεσίες και σελίδες JSP δείτε τα κεφάλαια
  // σχετικά με τη JDBC.
  private static CatalogItem[] items =
    { new CatalogItem
      ("hall001",
       "<I>Core Servlets and JavaServer Pages " +
         "2nd Edition</I> (Volume 1)" +
         " by Marty Hall and Larry Brown",
       "The definitive reference on servlets " +
         "and JSP from Prentice Hall and \n" +
         "Sun Microsystems Press.<P>Nominated for " +
         "the Nobel Prize in Literature.",
       39.95),
      new CatalogItem
        ("hall002",
         "<I>Core Web Programming, 2nd Edition</I> " +
           "by Marty Hall and Larry Brown",
         "One stop shopping for the Web programmer. " +
           "Topics include \n" +
           "<UL><LI>Thorough coverage of Java 2; " +
           "including Threads, Networking, Swing, \n" +
           "Java 2D, RMI, JDBC, and Collections\n" +
           "<LI>A fast introduction to HTML 4.01, " +
           "including frames, style sheets, and layers.\n" +
           "<LI>A fast introduction to HTTP 1.1, " +
           "servlets, and JavaServer Pages.\n" +
           "<LI>A quick overview of JavaScript 1.2\n" +
           "</UL>",
         49.99),
      new CatalogItem
        ("lewis001",
         "<I>The Chronicles of Narnia</I> by C.S. Lewis",
           "The classic children's adventure pitting " +
           "Aslan the Great Lion and his followers\n" +
           "against the White Witch and the forces " +
           "of evil. Dragons, magicians, quests, \n" +
           "and talking animals wound around a deep " +
           "spiritual allegory. Series includes\n" +
           "<I>The Magician's Nephew</I>,\n" +
           "<I>The Lion, the Witch and the Wardrobe</I>,\n" +
           "<I>The Horse and His Boy</I>,\n" +
           "<I>Prince Caspian</I>,\n" +
           "<I>The Voyage of the Dawn Treader</I>,\n" +
```

Λίστα 9-12 Catalog.java *(συνέχεια)*

```
          "<I>The Silver Chair</I>, and \n" +
          "<I>The Last Battle</I>.",
        19.95),
      new CatalogItem
        ("alexander001",
         "<I>The Prydain Series</I> by Lloyd Alexander",
          "Humble pig-keeper Taran joins mighty " +
          "Lord Gwydion in his battle against\n" +
          "Arawn the Lord of Annuvin. Joined by " +
          "his loyal friends the beautiful princess\n" +
          "Eilonwy, wannabe bard Fflewddur Fflam," +
          "and furry half-man Gurgi, Taran discovers " +
          "courage, nobility, and other values along\n" +
          "the way. Series includes\n" +
          "<I>The Book of Three</I>,\n" +
          "<I>The Black Cauldron</I>,\n" +
          "<I>The Castle of Llyr</I>,\n" +
          "<I>Taran Wanderer</I>, and\n" +
          "<I>The High King</I>.",
        19.95),
      new CatalogItem
        ("rowling001",
         "<I>The Harry Potter Series</I> by J.K. Rowling",
         "The first five of the popular stories " +
          "about wizard-in-training Harry Potter\n" +
          "topped both the adult and children's " +
          "best-seller lists. Series includes\n" +
          "<I>Harry Potter and the Sorcerer's Stone</I>,\n" +
          "<I>Harry Potter and the Chamber of Secrets</I>,\n" +
          "<I>Harry Potter and the " +
          "Prisoner of Azkaban</I>,\n" +
          "<I>Harry Potter and the Goblet of Fire</I>, and\n" +
          "<I>Harry Potter and the "+
          "Order of the Phoenix</I>.\n",
        59.95)
        };

  public static CatalogItem getItem(String itemID) {
    CatalogItem item;
    if (itemID == null) {
      return(null);
    }
    for(int i=0; i<items.length; i++) {
      item = items[i];
      if (itemID.equals(item.getItemID())) {
        return(item);
      }
    }
    return(null);
  }
}
```