Java Servlet i Java Server Pages

Autor: Marty Hall
Tłumaczenie: Piotr Rajca
ISBN: 83-7197-603-8
Tytuł oryginału: Core Servlets and JavaServer Pages
Format: B5, stron: 530

Niniejsza książka przeznaczona jest dla prawdziwych programistów. Jej celem nie jest przedstawienie potencjału internetowego handlu ani sposobów, w jaki aplikacje internetowe mogą zrewolucjonizować przedsiębiorstwo. Jest to praktyczna książka przeznaczona dla programistów, którzy już doskonale rozumieją, jak istotne jest tworzenie dynamicznych witryn WWW. Zadaniem niniejszej książki jest przedstawienie poprawnego sposobu prowadzenia takiej pracy. Prezentując metody tworzenia dynamicznych witryn WWW autor starał się uwzględnić najważniejsze używane techniki i opisać najczęściej napotykane problemy. Jednocześnie przedstawiono bardzo dużo praktycznych zastosowań (na przykład – ponad sto klas Javy). Podano szczegółowe przykłady dla wszystkich najważniejszych i najczęściej wykorzystywanych możliwości, a także zamieszczono podsumowania przedstawiające również rzadziej wykorzystywane zastosowania i wskazano dostępne w Internecie źródła informacji dotyczące interfejsów programistycznych (API) umożliwiających wykorzystanie tych najrzadziej stosowanych możliwości.

Nie ma tu wielu pobieżnych opisów licznych technologii. Nie było zamierzeniem autora, aby niniejsza książka stała się ostatecznym źródłem informacji na wszystkie omawiane w niej tematy. Są to zagadnienia bardzo obszerne. Przykładowo, istnieje kilka książek o porównywalnej objętości poświęconych wyłącznie JDBC. Należy podkreślić, że poszczególne zagadnienia zostały tutaj przedstawione na tyle szczegółowo, aby Czytelnik mógł samodzielnie rozpocząć tworzenie programów nadających się do praktycznego zastosowania. Jedynym warunkiem jest znajomość podstaw języka Java: jego zastosowania i wykorzystywania. W razie braku tych umiejętności Czytelnik powinien zaznajomić się z zasadami programowania w tym języku.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 9.

Śledzenie sesji

W niniejszym rozdziale przedstawiono sposób, w jaki można wykorzystać mechanizmy śledzenia sesji dostępne w serwletach w celu przechowywania informacji o użytkownikach poruszających się po witrynie WWW.

Potrzeba śledzenia sesji

HTTP jest protokołem „bezstanowym" — oznacza to, że za każdym razem, gdy przeglądarka pobiera stronę WWW, tworzone jest niezależne połączenie z serwerem. Serwer nie przechowuje automatycznie żadnych kontekstowych informacji skojarzonych z przeglądarką, która to połączenie nawiązała. Nawet w przypadku serwerów, które wykorzystują trwałe połączenia HTTP i za pomocą jednego połączenia obsługują wiele żądań zgłaszanych w niewielkich odstępach czasu, nie ma żadnych wbudowanych mechanizmów ułatwiających przechowywanie takich kontekstowych informacji. Brak takiego mechanizmu może powodować liczne problemy. Szczególnie widoczne są one w przypadku sklepów internetowych, gdzie istnieje oczywista potrzeba śledzenia sesji użytkownika.

Znane są trzy podstawowe sposoby rozwiązywania problemów tego typu — cookies, przepisywanie adresów URL oraz zastosowanie ukrytych pól formularzy.

Cookies

Istnieje możliwość wykorzystywania cookies w celu przechowywania informacji o sesji w sklepie internetowym, a każde kolejne połączenie z tym sklepem może powodować odszukanie danej sesji i pobranie przechowywanych na serwerze informacji o niej. Na przykład, utworzony w tym celu serwlet mógłby przeprowadzać następujące czynności:

```
String idSesji = utworzUnikalnyIdentyfikatorString();
Hashtable infoSesji = new Hashtable();
Hashtable tablicaGlobalna = odszukajTabliceZDanymiSesji();
```

```
tablicaGlobalna.put(idSesji, infoSesji);
Cookie cookieSesji = new Cookie("SESSIONID", idSesji);
sessionCookie.setPath("/");
response.addCookie(cookieSesji);
```

Dzięki temu podczas obsługi kolejnych żądań, na podstawie wartości cookie `"SESSIONID"`, serwer może pobierać z tablicy `tablicaGlobalna` tablicę asocjacyjną `infoSesji` zawierającą informacje o konkretnej sesji. Jest to bardzo dobre rozwiązanie, które często jest wykorzystywane przy obsłudze sesji. Niemniej jednak wciąż istnieje potrzeba posiadania narzędzi programistycznych wyższego poziomu, które pozwoliłyby na implementację bardziej wyrafinowanych rozwiązań. Choć serwlety dysponują łatwymi w obsłudze narzędziami wysokiego poziomu przeznaczonymi do obsługi cookies (opisano je w rozdziale 8.), wciąż jednak istnieje stosunkowo dużo drobnych czynności, które należy przeprowadzić samodzielnie. Do czynności tych można zaliczyć:

- pobieranie cookie przechowującego identyfikator sesji (możliwe jest nadesłanie kilku cookies z jednej przeglądarki);
- określenie odpowiedniego czasu wygaśnięcia ważności cookie (sesje, które są nieaktywne przez ponad 24 godziny prawdopodobnie powinny zostać zamknięte);
- skojarzenie tablic asocjacyjnych z każdym żądaniem;
- generowanie unikalnego identyfikatora sesji.

Należy podkreślić, że ze względu na doskonale znane i oczywiste zagrożenia prywatności, jakie wiążą się z wykorzystywaniem cookies (patrz podrozdział rozdziału 8. „Niektóre problemy związane ze stosowaniem cookies"), niektórzy użytkownicy wyłączają ich obsługę. Wynika z tego potrzeba posiadania, oprócz protokołu wysokiego poziomu, także alternatywnego rozwiązania tego problemu.

Przepisywanie adresów URL

Rozwiązanie to polega na dopisywaniu na końcu adresu URL pewnych dodatkowych, identyfikujących sesję danych, które pozwalają serwerowi na skojarzenie przekazanego identyfikatora z przechowywanymi informacjami dotyczącymi danej sesji. Na przykład, w adresie URL *http://komputer/katalog/plik.html;sessionid=1234* dołączone informacje o sesji mają postać *sessionid=1234*. Rozwiązanie to jest bardzo funkcjonalne, a przy tym posiada tę dodatkową zaletę, że można z niego korzystać nawet w przypadku przeglądarek nie obsługujących cookies oraz wtedy, gdy użytkownik wyłączył obsługę cookies. Jednak także i to rozwiązanie zmusza program działający na serwerze do wykonywania wielu prostych, lecz uciążliwych czynności. Poza tym należy zwracać szczególną uwagę, aby każdy adres URL odwołujący się do danej witryny i przekazywany do przeglądarki użytkownika (nawet w sposób niejawny, przykładowo, za pomocą nagłówka `Location`), został uzupełniony do dodatkowe informacje. Inną niedogodnością rozwiązania polegającego na przepisywaniu adresów URL jest fakt, że w po zakończeniu sesji i późniejszym nawiązania kolejnej sesji na danej witrynie za pomocą zapamiętanego adresu lub połączenia, wszystkie dotyczące jej informacje są tracone.

Ukryte pola formularzy

W formularzach HTML można umieszczać pola o następującej postaci:

```
<INPUT TYPE="HIDDEN" NAME="sesja" VALUE="...">
```

Wykorzystanie takiego elementu oznacza, że podczas przesyłania formularza podana nazwa elementu oraz jego wartość są dołączane do pozostałych informacji przekazywanych na serwer. Więcej informacji na temat ukrytych pól formularzy znajduje się w podrozdziale rozdziału 16. „Pola ukryte". W takich ukrytych polach formularzy można przechowywać informacje na temat sesji. Jednak wykorzystywanie tej metody ma jedną podstawową wadę — każda strona witryny musi być generowana dynamicznie.

Śledzenie sesji w serwletach

Serwlety udostępniają bardzo dobre rozwiązanie problemu śledzenia sesji — interfejs `HttpSession`. Możliwości funkcjonalne tego interfejsu bazują na wykorzystywaniu cookies lub przepisywania adresów URL. W rzeczywistości większość serwerów wykorzystuje cookies, pod warunkiem, że są one obsługiwane przez przeglądarkę użytkownika. Jeśli przeglądarka nie obsługuje cookies lub gdy ich obsługa została jawnie wyłączona, często automatycznie stosowane jest przepisywanie adresów URL. W ten sposób autorzy serwletów nie muszą jawnie obsługiwać cookies ani informacji dołączanych do adresów URL, a zyskują bardzo wygodny sposób przechowywania dowolnych obiektów skojarzonych z każdą sesją.

Narzędzia programistyczne do śledzenia sesji

Wykorzystanie śledzenia sesji w serwletach jest bardzo proste i wiąże się z pobieraniem obiektu sesji skojarzonego z obsługiwanym żądaniem, w razie konieczności z tworzeniem nowego obiektu sesji, pobieraniem informacji skojarzonych z sesją, zapisywaniem informacji w obiekcie sesji oraz z usuwaniem przerwanych lub zakończonych sesji. Dodatkowo, jeśli do użytkownika przekazywane są jakiekolwiek adresy URL odwołujące się do danej witryny i jeśli jest stosowane przepisywanie adresów URL, to do każdego adresu URL należy dołączać informacje o sesji.

Pobieranie obiektu HttpSession skojarzonego z bieżącym żądaniem

Obiekt `HttpSession` można pobierać za pomocą metody `getSession` interfejsu `HttpServletRequest`. System pobiera identyfikator użytkownika z cookie lub z informacji dołączonych do adresu URL w niewidoczny sposób, a następnie używa go jako klucza przy pobieraniu odpowiedniego elementu z tablicy utworzonych wcześniej obiektów

HttpSession. Wszystkie te czynności są wykonywane w sposób niezauważalny dla programisty, który jedynie wywołuje metodę getSession. Jeśli wywołanie tej metody zwraca wartość null, oznacza to, że z danym użytkownikiem jeszcze nie skojarzono żadnej sesji, a zatem można ją utworzyć. Sesje są tworzone bardzo często, zatem dostępna jest specjalna opcja pozwalająca na utworzenie nowej sesji, jeśli sesja dla danego użytkownika jeszcze nie istnieje. W tym celu wystarczy przekazać wartość true w wywołaniu metody getSession. A zatem, pierwszy krok przy rozpoczynaniu sesji zazwyczaj wygląda w następujący sposób:

```
HttpSession sesja = request.getSession(true);
```

Sprawdzenie, czy dana sesja istniała już wcześniej, lub czy też została właśnie utworzona, jest możliwe za pomocą metody isNew.

Pobieranie informacji skojarzonych z sesją

Obiekty HttpSession są przechowywane na serwerze. Są one automatycznie kojarzone z żądaniami podczas wykorzystywania niewidocznych mechanizmów, takich jak cookies lub przepisywanie adresów URL. Obiekty te posiadają wbudowaną strukturę danych, która umożliwia im przechowywanie dowolnej liczby kluczy oraz skojarzonych z nimi wartości. W specyfikacjach Java Servlet 2.1 oraz wcześniejszych do pobrania wartości uprzednio zapisanej w sesji stosuje się wywołanie session.getValue ("nazwaAtrybutu"). Metoda getValue zwraca obiekt typu Object, a zatem przywrócenie oryginalnego typu danej zapisanej w sesji jest możliwe poprzez zastosowanie odpowiedniego rzutowania typów. Metoda zwraca wartość null, jeśli nie ma atrybutu o podanej nazwie. Oznacza to, że przed wywołaniem jakichkolwiek metod obiektu skojarzonego z sesją należy sprawdzić, czy nie jest to wartość null.

W specyfikacji Java Servlet 2.2 metoda getValue została uznana za przestarzałą i zastąpiono ją metodą getAttribute. Nazwa tej metody lepiej odpowiada metodzie setAttribute (w specyfikacji 2.1 metodzie getValue odpowiadała metoda putValue, a nie setValue). W przykładach przedstawionych w niniejszej książce zastosowano metodę getValue, gdyż nie wszystkie dostępne na rynku komercyjne mechanizmy obsługi serwletów są zgodne ze specyfikacją 2.2.

Poniżej podano przykład wykorzystania mechanizmu śledzenia sesji. Założono, że ShoppingCart jest klasą zdefiniowaną w celu przechowywania informacji o zamawianych produktach (implementację tej klasy zamieszczono w podrozdziale „Internetowy sklep wykorzystujący koszyki i śledzenie sesji").

```
HttpSession sesja = request.getSession(true);
ShoppingCart koszyk = (ShoppingCart)sesja.getValue("shoppingCart");
if (koszyk == null) { // w sesji nie ma jeszcze koszyka
  koszyk = new ShoppingCart();
  sesja.putValue("shoppingCart",koszyk);
}
zrobCosZKoszykiem(koszyk);
```

W większości przypadków nazwa atrybutu jest znana, a celem działania jest pobranie wartości skojarzonej z tą nazwą. Istnieje także możliwość pobrania nazw wszystkich atrybutów skojarzonych z daną sesją — służy do tego metoda getValueNames zwracająca

tablicę łańcuchów znaków. Ta metoda jest jedynym sposobem określenia nazw atrybutów w przypadku korzystania z mechanizmów obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1. W przypadku mechanizmów zgodnych ze specyfikacją 2.2 nazwy atrybutów można pobierać za pomocą metody `getAttributeNames`. Metoda ta działa w bardziej spójny sposób, gdyż zwraca obiekt `Enumeration`, podobnie jak metody `getHeaderNames` oraz `getParameterNames` interfejsu `HttpServletRequest`.

Najczęściej wykorzystywanymi danymi są dane bezpośrednio skojarzone z sesją, niemniej jednak dostępne są także inne, użyteczne informacje dotyczące sesji. Poniżej przedstawiono metody interfejsu `HttpSession`:

public Object getValue(String nazwa)
public Object getAttribute(String nazwa)

Metody te pobierają z obiektu sesji wartość, która uprzednio została w nim zapisana. Obydwie metody zwracają wartość `null`, jeśli z podaną nazwą nie jest skojarzona żadna wartość. W mechanizmach obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1 należy używać metody `getValue`. W mechanizmach zgodnych ze specyfikacją 2.2 można używać obydwu metod, jednak zalecane jest stosowanie metody `getAttribute`, gdyż metoda `getValue` została uznana za przestarzałą.

public void putValue(String nazwa, Object wartość)
public void setAttribute(String nazwa, Object wartość)

Te metody kojarzą wartość atrybutu z jego nazwą. W mechanizmach obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1 należy używać metody `putValue`, natomiast w mechanizmach zgodnych ze specyfikacją 2.2 można stosować obydwie metody (rekomendowana jest metoda `setAttribute`, gdyż metoda `putValue` została uznana za przestarzałą). Jeśli obiekt przekazany w wywołaniu metody `putValue` lub `setAttribute` implementuje interfejs `HttpSessionBindingListener`, to po zapisaniu tego obiektu w sesji jest wywoływana jego metoda `valueBound`. Podobnie, jeśli obiekt implementuje interfejs `HttpSessionBindingListener`, po jego usunięciu z sesji jest wywoływana metoda `valueUnbound` tego obiektu.

public void removeValue(String nazwa)
public void removeAttribute(String nazwa)

Obydwie te metody powodują usunięcie wartości skojarzonych z podaną nazwą. Jeśli usuwana wartość implementuje interfejs `HttpSessionBindingListener`, wywoływana jest jej metoda `valueUnbound`. W mechanizmach obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1 należy używać metody `removeValue`. W mechanizmach zgodnych ze specyfikacją 2.2 preferowane jest użycie metody `removeAttribute`, choć w celu zapewnienia zgodności z poprzednimi wersjami oprogramowania można także używać metody `removeValue` (uznawanej za przestarzałą).

public String[] getValueNames()
public Enumeration getAttributeNames()

Te metody zwracają nazwy wszystkich atrybutów w danej sesji. W przypadku mechanizmów obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1 należy używać metody `getValueNames`. W przypadku mechanizmów zgodnych ze specyfikacją 2.2 metoda ta jest wciąż dostępna, lecz uznana za przestarzałą i z tego względu należy stosować metodę `getAttributeNames`.

public String getId()

Jest to metoda zwracająca unikalny identyfikator generowany dla każdej z sesji. Identyfikator ten jest czasami używany jako nazwa klucza. Dotyczy to sytuacji, gdy z sesją jest skojarzona tylko jedna wartość lub gdy informacje o sesji są rejestrowane.

public boolean isNew()

Jest to metoda zwracająca wartość `true`, jeśli klient jeszcze nigdy nie korzystał z sesji (zazwyczaj dlatego, że sesja została właśnie utworzona, a odbierane żądania jeszcze się do niej nie odwoływały). Jeśli dana sesja już istnieje od jakiegoś czasu, metoda ta zwraca wartość `false`.

public long getCreationTime()

Ta metoda zwraca czas utworzenia sesji wyrażony jako liczba milisekund, jakie upłynęły od początku 1970 roku (GMT). Aby przekształcić tę wartość do powszechnie stosowanej formy, należy przekazać ją do konstruktora klasy `Date` lub posłużyć się metodą `setTimeInMilis` klasy `GregorianCalendar`.

public long getLastAccessTime()

Metoda ta zwraca czas ostatniego przesłania sesji z przeglądarki na serwer. Czas ten jest wyrażony jako liczba milisekund, jakie upłynęły od początku 1970 roku (GMT).

public int getMaxInactiveInterval()
public void setMaxInactiveInterval(int ilośćSekund)

Pierwsza z tych metod pobiera, a druga określa czas (wyrażony w sekundach), w którym klient musi nadesłać żądanie, aby sesja nie została automatycznie unieważniona. Jeśli w wywołaniu metody `setMaxInactiveInterval` jest podana wartość mniejsza od zera, dana sesja nigdy nie zostanie automatycznie unieważniona. Należy pamiętać, iż limit czasu oczekiwania sesji jest przechowywany na serwerze i nie odpowiada dacie wygaśnięcia ważności cookies, która jest przesyłana do przeglądarki.

public void invalidate()

Wywołanie tej metody powoduje unieważnienie sesji i usunięcie z niej wszystkich skojarzonych z nią obiektów.

Kojarzenie informacji z sesją

Zgodnie z tym, co podano w poprzednim podrozdziale, informacje skojarzone z sesją można odczytywać za pomocą metody `getValue` (stosowanej w mechanizmach obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1) oraz `getAttribute` (stosowanej w mechanizmach obsługi serwletów zgodnych ze specyfikacją 2.2). Aby podać te informacje w przypadku korzystania z mechanizmów obsługi serwletów zgodnych ze specyfikacją 2.1, należy posłużyć się metodą `putValue`, podając w jej wywołaniu klucz oraz wartość. W przypadku wykorzystywania mechanizmów zgodnych ze specyfikacją Java Servlet 2.2, należy zastosować metodę `setAttribute`. Nazwa tej metody jest bardziej spójna z ogólnie stosowanym nazewnictwem, gdyż występują tu notacje `set/get` stosowane w komponentach JavaBeans. Aby wartości przechowywane w sesji mogły dawać jakieś efekty w chwili ich zapisywania w sesji, używane obiekty muszą implementować interfejs `HttpSessionBindingListener`. Dzięki temu za każdym razem, gdy za pomocą metody `putValue` lub `setAttribute` nastąpi skojarzenie jakiegoś obiektu z sesją, zostanie wywołana jego metoda `valueBound`.

Trzeba pamiętać, że metody `putValue` oraz `setAttribute` zastępują wszelkie wartości skojarzone z podaną nazwą. Jeśli trzeba usunąć tę wartość bez podawania zamiennika, należy posłużyć się metodą `removeValue` (w przypadku korzystania z mechanizmów obsługi serwletów zgodnych ze specyfikacją Java Servlet 2.1) lub `removeAttribute` (w przypadku korzystania z mechanizmów obsługi serwletów zgodnych ze specyfikacją 2.2). Jeśli usuwane obiekty implementują interfejs `HttpSessionBindingListener`, wywołanie którejkolwiek z tych metod spowoduje wywołanie metody `valueUnbound` usuwanego obiektu. Czasami zachodzi potrzeba jedynie zastąpienia istniejącej wartości atrybutu. W takim przypadku można zastosować metodę przedstawioną w drugim wierszu poniższego przykładu (określającego wartość atrybutu `"referringPage"`). Jeżeli należy pobrać aktualną wartość atrybutu i zmodyfikować ją można posłużyć się przedstawionym w poniższym przykładzie sposobem (operacje wykonywane na atrybucie `"previousItem"`). Założono, że w poniższym przykładzie wykorzystywane są dwie klasy — klasa `ShoppingCart` dysponująca metodą `addItem`, obiekty której służą do przechowywania informacji o wybranych produktach, oraz klasa `Catalog` udostępniająca statyczną metodę `getItem` zwracającą produkt na podstawie podanego identyfikatora. Implementacje tych klas zamieszczono w podrozdziale „Internetowy sklep wykorzystujący koszyki i śledzenie sesji”.

```
HttpSession sesja = request.getSession(true);
sesja.putValue("referringPage", request.getHeader("Referer"));
ShoppingCart koszyk =
    (ShoppingCart) sesja.getValue("previousItem");
if (koszyk == null) { // w sesji nie ma jeszcze żadnego koszyka
  koszyk = new ShoppingCart();
  sesja.putValue("previousItem", koszyk);
}
String idProduktu = request.getParameter("itemID");
if (idProduktu != null) {
  koszyk.addItem(Catalog.getItem(idProduktu));
}
```

Kończenie sesji

Sesja automatycznie staje się nieaktywna, jeśli odstęp pomiędzy kolejnymi żądaniami przekroczy czas określony przez metodę `getMaxInactiveInterval`. W tym momencie automatycznie są usuwane wszelkie skojarzenia obiektów z tą sesją. To z kolei sprawia, że obiekty implementujące interfejs `HttpSessionBindingListener` są zawiadomiane o fakcie usunięcia ich z sesji.

Istnieje również sposób jawnego kończenia sesji — służy do tego metoda `invalidate`.

Przepisywanie adresów URL przesyłanych do przeglądarki

Jeśli przy śledzeniu sesji wykorzystywana jest metoda przepisywania adresów URL i jeśli do przeglądarki przesyłane są adresy URL odwołujące się do danej witryny, konieczne jest jawne dodawanie do nich informacje o sesji. Wspomniano, że serwlety mogą automatycznie stosować metodę przepisywania adresów URL, jeśli przeglądarka nie obsługuje cookies. Dlatego też należy zawsze kodować wszystkie adresy URL odwołujące się do witryny. Adresy URL odwołujące się do tej samej witryny mogą być stosowane w dwóch przypadkach. Pierwszym z nich są strony WWW generowane przez serwlety. Adresy umieszczane na takich stronach powinny być odpowiednio zakodowane za pomocą metody `encodeURL` interfejsu `HttpServletResponse`. Metoda ta automatycznie stwierdza, czy przepisywanie adresów jest aktualnie wykorzystywane i w razie konieczności dopisuje do adresu informacje o sesji. W przeciwnym razie metoda zwraca adresu URL w oryginalnej postaci.

Oto przykład:

```
String oryginalnyURL = jakisURL_wzglednyLubBezwzgledny;
String zakowowanyURL = response.encodeURL(oryginalnyURL);
out.println("<A HREF=\"" + zakodowanyURL + "\">...</A>);
```

Drugim przypadkiem, gdzie podaje się adresy URL odwołujące się do tej samej witryny, jest wywołanie metody `sendRedirect` (oznacza to, że adresy te są umieszczane w nagłówku odpowiedzi `Location`). W tym przypadku obowiązują inne sposoby określania, czy do adresu należy dodać informacje o sesji czy nie, a zatem nie można posłużyć się metodą `encodeURL`. Interfejs `HttpServletResponse` udostępnia metodę `encodeRedirectURL`, którą można zastosować w takich sytuacjach. Oto przykład zastosowania tej metody:

```
String oryginalnyURL = jakisURL; // specyfikacja 2.2 pozwala na używanie
                                 // względnych adresów URL
String zakodowanyURL = response.encodeRedirectURL(oryginalnyURL);
response.sendRedirect(zakodowanyURL);
```

Często się zdarza, że w chwili tworzenia serwletu nie można przewidzieć, czy w przyszłości stanie się on częścią grupy strony wykorzystujących śledzenie sesji. Warto zatem uwzględnić taką możliwość i zapewnić kodowanie w serwlecie wszystkich adresów URL odwołujących się do tej samej witryny.

Wskazane jest, aby wszystkie adresy URL odwołujące się do tej samej witryny przekształcać za pomocą metod `response.encodeURL` lub `response.encodeRedirectURL`, niezależnie od tego, czy serwlet wykorzystuje mechanizmy śledzenia sesji.

Serwlet generujący indywidualny licznik odwiedzin dla każdego użytkownika

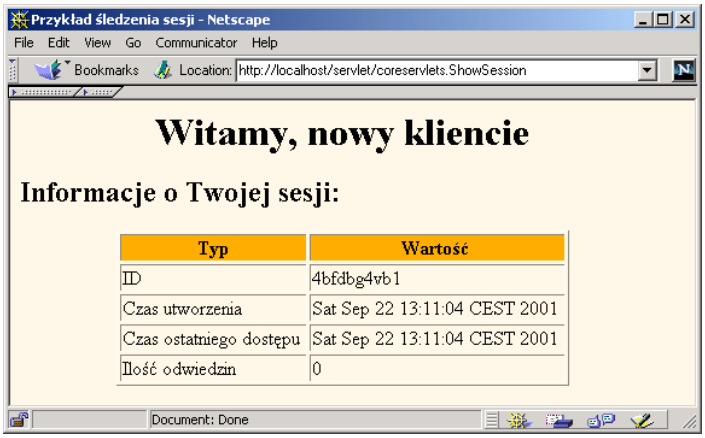
Listing 9.1 przedstawia prosty serwlet wyświetlający podstawowe informacje dotyczące sesji. Po nawiązaniu połączenia serwlet pobiera istniejącą sesję lub, jeśli sesji jeszcze nie ma, tworzy ją. Obydwie te czynności realizowane są za pomocą jednego wywołania — `request.getSession(true)`. Następnie serwlet sprawdza, czy w sesji zapisano atrybut typu `Integer` o nazwie „`accessCount`". Jeśli atrybut ten nie jest odnaleziony, zakłada się, że liczba wcześniejszych odwiedzin danej strony wynosi `0`. Jeśli atrybut `accessCount` zostanie odnaleziony, serwlet pobiera jego wartość. Wartość ta jest następnie inkrementowana i ponownie kojarzona z sesją poprzez wywołanie metody `putValue`. Następnie serwlet generuje niewielką tabelę HTML zawierającą informacje o sesji. Rysunek 9.1 przedstawia stronę wygenerowaną przez serwlet podczas pierwszej wizyty użytkownika na witrynie, a rysunek 9.2 — po jej kilkukrotnym odświeżeniu.

Listing 9.1. *ShowSession.java*

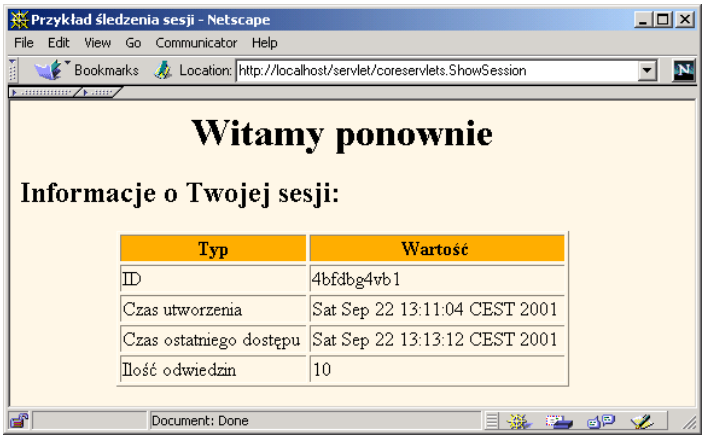
```
package coreservlets;

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

/** Prosty przykład śledzenia sesji. Bardziej
 *  zaawansowanym przykładem jest implementacja koszyków
 *  ShoppingCart.java.
 */

public class ShowSession extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html; charset=ISO-8859-2");
    PrintWriter out = response.getWriter();
    String title = "Przykład śledzenia sesji";
    HttpSession session = request.getSession(true);
    String heading;
    /* W mechanizmach obsługi serwletów zgodnych ze specyfikacją
     * Java Servlet 2.2 zamiast metody getValue należy stosować
```

```
      * metodę getAttribute.
      */
     Integer accessCount =
       (Integer)session.getValue("accessCount");
     if (accessCount == null) {
       accessCount = new Integer(0);
       heading = "Witamy, nowy kliencie";
     } else {
       heading = "Witamy ponownie";
       accessCount = new Integer(accessCount.intValue() + 1);
     }
     /* W mechanizmach obsługi serwletów zgodnych ze specyfikacją
      * Java Servlet 2.2 zamiast metody putValue należy stosować
      * metodę setAttribute.
      */
     session.putValue("accessCount", accessCount);

     out.println(ServletUtilities.headWithTitle(title) +
                 "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                 "<H1 ALIGN=\"CENTER\">" + heading + "</H1>\n" +
                 "<H2>Informacje o Twojej sesji:</H2>\n" +
                 "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                 "<TR BGCOLOR=\"#FFAD00\">\n" +
                 "  <TH>Typ<TH>Wartość\n" +
                 "<TR>\n" +
                 "  <TD>ID\n" +
                 "  <TD>" + session.getId() + "\n" +
                 "<TR>\n" +
                 "  <TD>Czas utworzenia\n" +
                 "  <TD>" +
                 new Date(session.getCreationTime()) + "\n" +
                 "<TR>\n" +
                 "  <TD>Czas ostatniego dostępu\n" +
                 "  <TD>" +
                 new Date(session.getLastAccessedTime()) + "\n" +
                 "<TR>\n" +
                 "  <TD>Ilość odwiedzin\n" +
                 "  <TD>" + accessCount + "\n" +
                 "</TABLE>\n" +
                 "</BODY></HTML>");

   }

   /** Żądania GET i POST mają być obsługiwane jednakowo. */

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

Rysunek 9.1. *Strona wygenerowana po pierwszym wykonaniu serwletu ShowSession*



Rysunek 9.2. *Strona wygenerowana po jedenastym wykonaniu serwletu ShowSession*



Internetowy sklep wykorzystujący koszyki i śledzenie sesji

W tym podrozdziale przedstawiono rozbudowany przykład opisujący sposób tworzenia internetowego sklepu wykorzystującego mechanizmy śledzenia sesji. W pierwszej części podrozdziału przedstawiono sposób tworzenia stron wyświetlających informacje o sprzedawanych produktach. Kod każdej ze stron prezentujących sprzedawane produkty zawiera wyłącznie tytuł strony oraz identyfikatory produktów, jakie mają się na niej pojawić. Sam kod HTML każdej z tych stron jest generowany automatycznie przez metody klasy bazowej na podstawie opisów produktów przechowywanych w katalogu. W drugiej części podrozdziału opisano stronę obsługującą zamawianie produktów. Strona ta kojarzy każdego z użytkowników z koszykiem i pozwala użytkownikowi na zmodyfikowanie liczby każdego z zamówionych produktów. W celu prawidłowego kojarzenia użytkowników z koszykami strona ta wykorzystuje mechanizmy

śledzenia sesji. W trzeciej części niniejszego podrozdziału przedstawiono implementację koszyka — struktury danych reprezentującej poszczególne produkty oraz zamówienia — oraz katalogu produktów.

Tworzenie interfejsu użytkownika

Listing 9.2 przedstawia abstrakcyjną klasę bazową używaną podczas tworzenia serwletów, które mają prezentować sprzedawane produkty. Serwlet ten pobiera identyfikatory produktów, odszukuje je w katalogu, a następnie pobiera z niego nazwy i ceny produktów i wyświetla je na stronie umożliwiającej zamawianie. Listingi 9.3 oraz 9.4 przedstawiają łatwy sposób tworzenia stron zawierających informacje o produktach za pomocą klasy bazowej z listingu 9.2. Wygląd stron zdefiniowanych na listingach 9.3 oraz 9.4 został przedstawiony na rysunkach 9.3 oraz 9.4.

Listing 9.2. *CatalogPage.java*

```
package coreservlets;

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

/** Klasa bazowa stron prezentujących produkty z katalogu.
 *  Serwlety dziedziczące po  tej klasy bazowej muszą
 *  określić wyświetlane elementy katalogu oraz tytuł
 *  strony <i>zanim</i> serwlet zostanie wykonany.
 *  W tym celu w metodzie init takiego serwletu należy
 *  wywołać metody setItems oraz setTitle klasy bazowej.
 */

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

  /** Dysponując tablicą identyfikatorów produktów odszukaj
   *  je w katalogu (Catalog) i zapisz odpowiadające im
   *  obiekty Item do tablicy items. Obiekty Item zawierają
   *  krótki opis, pełny opis oraz cenę produktu, a ich
   *  unikalnym kluczem jest identyfikator produktu.
   *  <P>
   *  Serwlety dziedziczące po klasie CatalogPage
   *  <b>muszą</b> wywoływać tę metodę (zazwyczaj
   *  w metodzie init) zanim serwlet zostanie wywołany.
   */

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

```
/** Określa tytuł strony, który jest wyświetlany na
   *  stronie wynikowej w nagłówku <H1>.
   *  <P>
   *  Serwlety dziedziczące po klasie CatalogPage
   *  <b>muszą</b> wywoływać tę metodę (zazwyczaj
   *  w metodzie init) zanim serwlet zostanie wywołany.
   */

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

  /** Metoda w pierwszej kolejności wyświetla tytuł, a następnie
   *  dla każdego produktu z katalogu, który ma być przedstawiony
   *  na danej stronie, wyświetla jego krótki opis (w nagłówku
   *  <H2>), cenę (w nawiasach) oraz pełny opis poniżej.
   *  Poniżej opisu każdego produktu wyświetlany jest przycisk umożliwiający
   *  złożenie zamówienia dotyczącego danego produktu - informacje
   *  przesyłane są do serwletu OrderPage.
   *  <P>
   *  Aby zobaczyć kod HTML generowany przez tę metodę, należy
   *  wykonać serwlet KidsBooksPage lub TechBooksPage (obydwie
   *  te klasy dziedziczą po abstrakcyjnej klasie CatalogPage)
   *  i wybrać opcję "Wyświetl kod źródłowy" (lub jej odpowiednik)
   *  w przeglądarce.
   */

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html; charset=ISO-8859-2");
    if (items == null) {
      response.sendError(response.SC_NOT_FOUND,
                         "Brak produktów.");
      return;
    }
    PrintWriter out = response.getWriter();
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
    Item item;
    for(int i=0; i<items.length; i++) {
      out.println("<HR>");
      item = items[i];
      // Wyświetl informacje o błędzie, jeśli klasa potomna
      // podała identyfikator produktu, którego nie ma w katalogu
      if (item == null) {
        out.println("<FONT COLOR=\"RED\">" +
                    "Nieznany identyfikator produktu" + itemIDs[i] +
                    "</FONT>");
      } else {
        out.println();
        String formURL =
          "/servlet/coreservlets.OrderPage";
        // Adresy URL odwołujące się do tej samej witryny należy
        // przekształcać za pomocą metody 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=\"Dodaj do koszyka\">\n" +
           "</CENTER>\n<P>\n</FORM>");
}
    }
    out.println("<HR>\n</BODY></HTML>");
  }

  /** Żądania POST i GET mają być obsługiwane tak samo. */

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

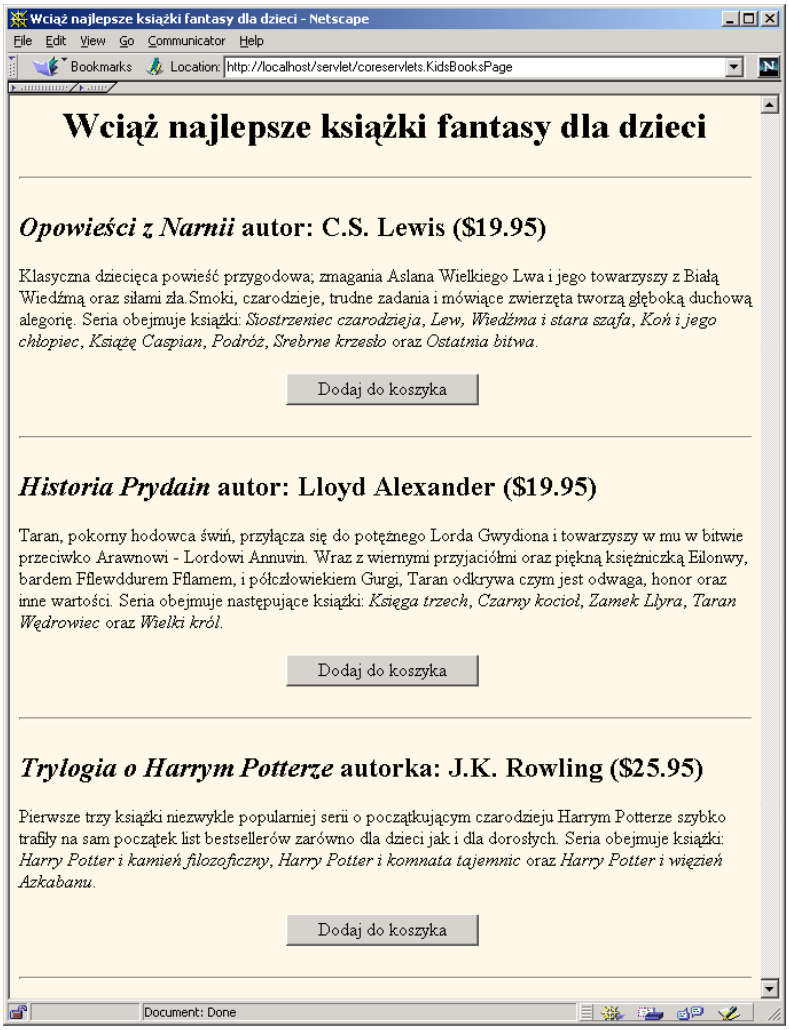
Listing 9.3. *KidsBooksPage.java*

```
package coreservlets;

/** Klasa potomna serwletu CatalogPage wyświetlająca
 *  stronę WWW umożliwiającą zamówienie trzech znanych
 *  serii książek dla dzieci.
 *  Zamówienia są przesyłane do serwletu OrderPage.
 */

public class KidsBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "lewis001", "alexander001", "rowling001" };
    setItems(ids);
    setTitle("Wciąż najlepsze książki fantasy dla dzieci");
  }
}
```

Listing 9.4. *TechBooksPage.java*

```
package coreservlets;

/** Klasa potomna serwletu CatalogPage wyświetlająca
 *  stronę WWW umożliwiającą zamówienie dwóch
 *  doskonałych książek komputerowych.
 *  Zamówienia są przesyłane do serwletu OrderPage.
 */

public class TechBooksPage extends CatalogPage {
  public void init() {
    String[] ids = { "hall001", "hall002" };
    setItems(ids);
```

```
      setTitle("Wciąż najlepsze książki komputerowe");
    }
  }
```

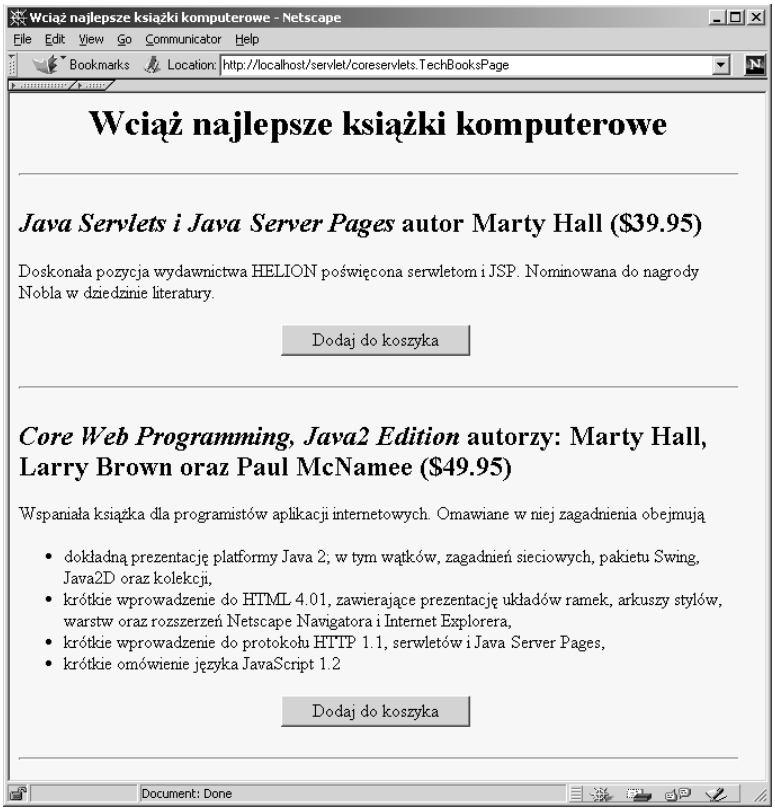


Rysunek 9.3. *Wyniki wykonania serwletu KidsBooksPage*

Obsługa zamówień

Listing 9.5 przedstawia kod serwletu służącego do obsługi zamówień nadsyłanych przez różne strony katalogowe (przedstawione w poprzedniej części podrozdziału). Serwlet ten kojarzy każdego użytkownika z koszykiem wykorzystując przy tym mechanizmy śledzenia sesji. Ponieważ każdy użytkownik dysponuje osobnym koszykiem, jest mało prawdopodobne, że wiele wątków będzie jednocześnie próbowało uzyskać dostęp do tego samego koszyka. Niemniej jednak można wyobrazić sobie sytuację, w których mógłby nastąpić jednoczesny dostęp do tego samego koszyka.

Rysunek 9.4. *Wyniki wykonania serwletu TechBooksPage*



Zatem w celu zapewnienia wysokiego poziomu bezpieczeństwa kod serwletu synchronizuje dostęp do koszyków na podstawie obiektu sesji. W ten sposób inne wątki korzystające z tej samej sesji nie będą mogły równocześnie uzyskiwać dostępu do przechowywanych w niej informacji, choć wciąż będzie możliwa równoczesna obsługa żądań nadsyłanych przez różnych użytkowników. Typowe wyniki wykonania tego serwletu zostały przedstawione na rysunkach 9.5 oraz 9.6.

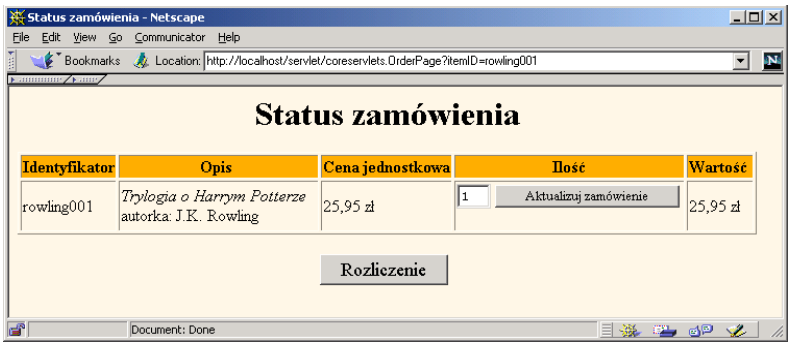
Listing 9.5. *OrderPage.java*

```
package coreservlets;

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

/** Wyświetla informacje o produktach, które aktualnie
 *  znajdują się w koszyku (ShoppingCart). Przeglądarki
 *  mają swoje własne sesje, na podstawie których
 *  określane jest przynależność koszyków. Jeśli to
 *  jest pierwsza wizyta na stronie umożliwiającej
 *  składanie zamówień, jest tworzony nowy koszyk.
 *  Użytkownicy zazwyczaj przechodzą na tę stronę ze stron
 *  prezentujących produkty, które można zamawiać, dlatego
```

```
 *  też ta strona dodaje nowy element do koszyka. Jednak
 *  użytkownicy mogą zapamiętać adres tej strony i wyświetlać
 *  ją posługując się listą ulubionych stron. Mogą także
 *  wrócić na nią klikając przycisk "Aktualizuj zamówienie"
 *  po zmianie liczby egzemplarzy jednego z zamawianych
 *  produktów.
 */

public class OrderPage extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    ShoppingCart cart;
    synchronized(session) {
      cart = (ShoppingCart)session.getValue("shoppingCart");
      // Dla nowych użytkowników tworzone są nowe koszyki.
      // Użytkownicy, którzy już dysponują koszykami
      // używają tych, które zostały dla nich wcześniej utworzone .
      if (cart == null) {
        cart = new ShoppingCart();
        session.putValue("shoppingCart", cart);
      }
      String itemID = request.getParameter("itemID");
      if (itemID != null) {
        String numItemsString =
          request.getParameter("numItems");
        if (numItemsString == null) {
          // Jeśli w żądaniu został podany identyfikator (ID) ale
          // nie liczba, to oznacza to, że użytkownik trafił
          // tutaj klikając przycisk "Dodaj do koszyka" na jednej
          // ze stron prezentującej produkty z katalogu.
          cart.addItem(itemID);
        } else {
          // Jeśli w żądaniu został podany zarówno identyfikator
          // (ID) jak i liczba, to oznacza to, że użytkownik
          // trafił na stronę klikając przycisk "Aktualizuj
          // zamówienie" po zmianie liczby egzemplarzy jednego
          // z zamawianych produktów. Należy zwrócić uwagę, iż podanie
          // wartości 0 jako liczby egzemplarzy zamawianego produktu
          // sprawi, że dany produkt zostanie usunięty z koszyka.
          int numItems;
          try {
            numItems = Integer.parseInt(numItemsString);
          } catch(NumberFormatException nfe) {
            numItems = 1;
          }
          cart.setNumOrdered(itemID, numItems);
        }
      }
    }
    // Pokaż status zamówienia niezależnie od tego, czy użytkownik
    // je zmodyfikował czy nie.
    response.setContentType("text/html; charset=ISO-8859-2");
    PrintWriter out = response.getWriter();
    String title = "Status zamówienia";
    out.println(ServletUtilities.headWithTitle(title) +
```

```
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=\"CENTER\">" + title + "</H1>");
synchronized(session) {
  Vector itemsOrdered = cart.getItemsOrdered();
  if (itemsOrdered.size() == 0) {
    out.println("<H2><I>Brak produktów w koszyku...</I></H2>");
  } else {
    // Jeśli w koszyku jest co najmniej jeden produkt,
    // wyświetl tabelę z informacjami o nim.
    out.println
      ("<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
       "<TR BGCOLOR=\"#FFAD00\">\n" +
       "  <TH>Identyfikator<TH>Opis\n" +
       "  <TH>Cena jednostkowa<TH>Ilość<TH>Wartość");
    ItemOrder order;

    // Zaokrąglamy do dwóch miejsc po przecinku,
    // wstawiamy znak dolara (lub innej waluty), itd.
    // wszystko zgodnie z bieżącymi ustawieniami lokalnymi.
    NumberFormat formatter =
      NumberFormat.getCurrencyInstance();

    String formURL =
      "/servlet/coreservlets.OrderPage";
    // Adresy URL odwołujące się do stron tej samej witryny
    // przekształcamy przy użyciu metody encodeURL.
    formURL = response.encodeURL(formURL);

    // Dla każdego produktu umieszczonego w koszyku
    // tworzymy wiersz tabeli zawierający identyfikator
    // produktu (ID), opis, jego cenę jednostkową,
    // liczbę zamówionych egzemplarzy oraz łączną cenę.
    // Liczbę zamawianych egzemplarzy wyświetlamy
    // w polu tekstowym, tak aby użytkownik mógł ją zmienić.
    // Dodatkowo, obok pola, wyświetlamy przycisk
    // "Aktualizuj zamówienie", który powoduje ponowne
    // przesłanie tej samej strony na serwer, przy czym
    // zmieniana jest liczba zamawianych egzemplarzy
    // danego produktu.
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.elementAt(i);
      out.println
        ("<TR>\n" +
         "  <TD>" + order.getItemID() + "\n" +
         "  <TD>" + order.getShortDescription() + "\n" +
         "  <TD>" +
         formatter.format(order.getUnitCost()) + "\n" +
         "  <TD>" +
         "<FORM ACTION=\"" + formURL + "\">\n" +
         "<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=\"Aktualizuj zamówienie\">\n" +
```

```
                "</SMALL>\n" +
                "</FORM>\n" +
                "  <TD>" +
                formatter.format(order.getTotalCost()));
            }
            String checkoutURL =
              response.encodeURL("/Checkout.html");
            // Pod tabelą wyświetlany jest przycisk "Rozliczenie"
            out.println
              ("</TABLE>\n" +
               "<FORM ACTION=\"" + checkoutURL + "\">\n" +
               "<BIG><CENTER>\n" +
               "<INPUT TYPE=\"SUBMIT\"\n" +
               "       VALUE=\"Rozliczenie\">\n" +
               "</CENTER></BIG></FORM>");
          }
          out.println("</BODY></HTML>");
        }
      }

      /** Żądania GET i POST są obsługiwane w identyczny sposób */

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

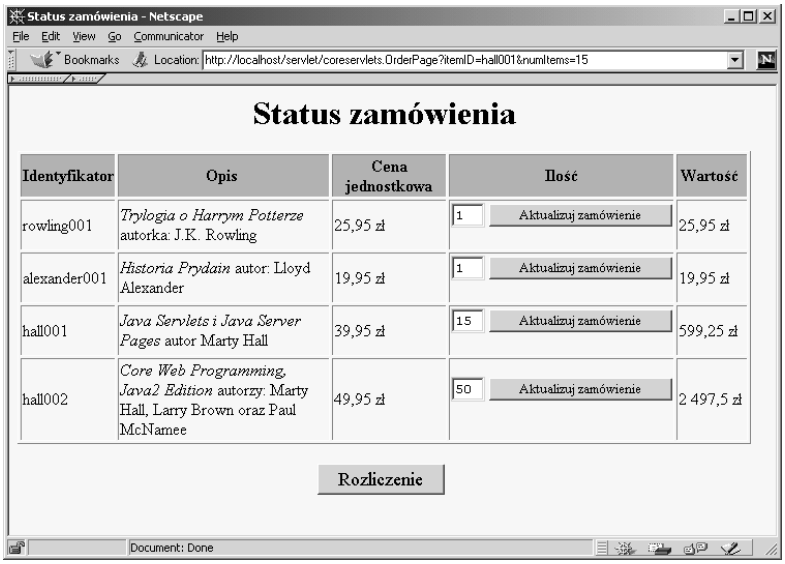


Rysunek 9.5. *Wyniki wygenerowane przez serwlet OrderPage po kliknięciu przycisku Dodaj do koszyka na stronie KidsBooksPage*

To czego nie widać: implementacja koszyka i katalogu produktów

Listing 9.6 przedstawia implementację koszyka. Obiekt koszyka — `ShoppingCart` — zawiera obiekt `Vector` przechowujący informacje o zamówionych produktach i dysponuje metodami umożliwiającymi dodawanie i aktualizację zamówienia. Na listingu 9.7 został przedstawiony kod obiektu reprezentującego element katalogu produktów. Klasa przedstawiona na listingu 9.8 reprezentuje status zamówienia konkretnego produktu. Wreszcie listing 9.9 przedstawia implementację katalogu produktów.

Rysunek 9.6. *Wyniki wygenerowane przez serwlet OrderPage po zamówieniu kilku dodatkowych książek i wprowadzeniu kilku zmian w zamówieniu*



Listing 9.6. *ShoppingCart.java*

```
package coreservlets;

import java.util.*;

/** Klasa implementująca koszyk - jest to struktura danych
 *  służąca do przechowywania informacji o zamawianych
 *  produktach.
 *  Serwlet OrderPage kojarzy jeden z tych koszyków
 *  z każdą sesją.
 */

public class ShoppingCart {
  private Vector itemsOrdered;

  /** Twórz pusty koszyk */

  public ShoppingCart() {
    itemsOrdered = new Vector();
  }

/** Zwraca Vector obiektów ItemOrder zawierających
   *  informacje o zamówionych produktach i ich liczbie.
   */

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

  /** Przegląda koszyk i sprawdza, czy znajduje się już
   *  w nim zamówienie dotyczące produktu o podanym
   *  identyfikatorze. Jeśli takie zamówienie zostanie odnalezione
   *  to liczba egzemplarzy danego produktu jest inkrementowana.
   *  Jeśli nie ma zamówienia dotyczącego podanego produktu,
```

```
   *  to metoda pobiera z katalogu (Catalog) informacje o nim
   *  i dodaje do koszyka odpowiedni obiekt.
   */

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

  /** Przegląda koszyk w poszukiwaniu wpisu dotyczącego
   *  produktu o podanym identyfikatorze. Jeśli podana liczba
   *  jest większa od zera, zostaje ona wykorzystana do określenia
   *  liczbie zamówionych egzemplarzy danego produktu. Jeśli
   *  przekazana liczba ma wartość 0 (lub mniejszą od zera,
   *  co może nastąpić w przypadku błędu użytkownika),
   *  wpis reprezentujący dany produkt jest usuwany z koszyka.
   */

  public synchronized void setNumOrdered(String itemID,
                                         int numOrdered) {
    ItemOrder order;
    for(int i=0; i<itemsOrdered.size(); i++) {
      order = (ItemOrder)itemsOrdered.elementAt(i);
      if (order.getItemID().equals(itemID)) {
        if (numOrdered <= 0) {
          itemsOrdered.removeElementAt(i);
        } else {
          order.setNumItems(numOrdered);
        }
        return;
      }
    }
    ItemOrder newOrder =
      new ItemOrder(Catalog.getItem(itemID));
    itemsOrdered.addElement(newOrder);
  }
}
```

Listing 9.7. *Item.java*

```
package coreservlets;

/** Opisuje element katalogu dla internetowego sklepu.
 *  identyfikator (itemID) w niepowtarzalny sposób identyfikuje każdy
 *  element, krótki opis (shortDescription) zawiera
 *  krótkie informacje o produkcie (takie jak nazwisko
 *  autora i tytuł książki), a długi opis (longDescription)
 *  to kilka zdań dokładniej opisujących dany produkt; w końcu
```

```
 *  cena (cost) jest jednostkową ceną produktu.
 *  Zarówno krótki, jak i długi opis może zawierać kod HTML.
 */

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

  public Item(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;
  }

  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;
  }
}
```

Listing 9.8. *ItemOrder.java*

```
package coreservlets;

/** Kojarzy element katalogu (Item) z konkretnym zamówieniem
 *  poprzez zapamiętanie informacji o liczbie zamawianych
 *  egzemplarzy danego produktu oraz ich łącznej wartości.
```

```
 *  Udostępnia także przydatne metody umożliwiające
 *  operowanie na informacjach przechowywanych w obiekcie Item
 *  bez konieczności jego pobierania.
 */

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

  public ItemOrder(Item item) {
    setItem(item);
    setNumItems(1);
  }

  public Item getItem() {
    return(item);
  }

  protected void setItem(Item 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());
  }
}
```

Listing 9.9. *Catalog.java*

```
package coreservlets;

/** Katalog zawierający informacje o produktach
 *  dostępnych w internetowym sklepie.
 */

public class Catalog {
  // Normalnie te informacje byłyby przechowywane i pobierane
  // z bazy danych.
  private static Item[] items =
    { new Item("hall001",
               "<I>Java Servlets i Java Server Pages</I> " +
                 " autor Marty Hall",
               "Doskonała pozycja wydawnictwa HELION poświęcona " +
                 "serwletom i JSP.\n" +
                 "Nominowana do nagrody Nobla w dziedzinie literatury.",
               39.95),
      new Item("hall002",
               "<I>Core Web Programming, Java2 Edition</I> " +
                 " autorzy: Marty Hall, Larry Brown oraz " +
                 "Paul McNamee",
               "Wspaniała książka dla programistów aplikacji " +
                 "internetowych. Omawiane w niej zagadnienia obejmują \n" +
                 "<UL><LI>dokładną prezentację platformy Java 2; " +
                 "w tym wątków, zagadnień sieciowych, pakietu Swing, \n" +
                 "Java2D oraz kolekcji,\n" +
                 "<LI>krótkie wprowadzenie do HTML 4.01, " +
                 "zawierające prezentację układów ramek, arkuszy stylów, \n" +
                 "warstw oraz rozszerzeń Netscape Navigator i " +
                 "Internet Explorer,\n" +
                 "<LI>krótkie wprowadzenie do protokołu HTTP 1.1, " +
                 "serwletów i Java Server Pages,\n" +
                 "<LI>krótkie omówienie języka JavaScript 1.2\n" +
                 "</UL>",
               49.95),
      new Item("lewis001",
               "<I>Opowieści z Narnii</I> autor: C.S. Lewis",
               "Klasyczna dziecięca powieść przygodowa; zmagania " +
                 "Aslana Wielkiego Lwa i jego towarzyszy\n" +
                 "z Białą Wiedźmą oraz siłami zła." +
                 "Smoki, czarodzieje, trudne zadania \n" +
                 "i mówiące zwierzęta tworzą głęboką duchową " +
                 "alegorię. Seria obejmuje książki\n" +
                 "<I>Siostrzeniec czarodzieja</I>,\n" +
                 "<I>Lew, Wiedźma i stara szafa</I>,\n" +
                 "<I>Koń i jego chłopiec</I>,\n" +
                 "<I>Książę Caspian</I>,\n" +
                 "<I>Podróż</I>,\n" +
                 "<I>Srebrne krzesło</I> oraz \n" +
                 "<I>Ostatnia bitwa</I>.",
               19.95),
      new Item("alexander001",
               "<I>Historia Prydain</I> autor: Lloyd Alexander",
               "Taran, pokorny hodowca świń, przyłącza się do " +
                 "potężnego Lorda Gwydiona i towarzyszy w mu w \n" +
```

```
                "bitwie przeciwko Arawnowi - Lordowi Annuvin. Wraz " +
                "z wiernymi przyjaciółmi oraz piękną księżniczką \n" +
                "Eilonwy, bardem Fflewddurem Fflamem, " +
                "i półczłowiekiem Gurgi, Taran odkrywa " +
                "czym jest odwaga, honor oraz inne wartości.\n" +
                "Seria obejmuje następujące książki: \n" +
                "<I>Księga trzech</I>, \n" +
                "<I>Czarny kocioł</I>, \n" +
                "<I>Zamek Llyra</I>, \n" +
                "<I>Taran Wędrowiec</I> oraz \n" +
                "<I>Wielki król</I>.",
              19.95),
      new Item("rowling001",
              "<I>Trylogia o Harrym Potterze</I> autorka: " +
                "J.K. Rowling",
              "Pierwsze trzy książki niezwykle popularniej serii o " +
                "początkującym czarodzieju Harrym Potterze \n" +
                "szybko trafiły na sam początek list bestsellerów " +
                "zarówno dla dzieci jak i dla dorosłych. Seria \n" +
                "obejmuje książki: \n" +
                "<I>Harry Potter i kamień " +
                  "filozoficzny</I>, \n" +
                "<I>Harry Potter i komnata " +
                  "tajemnic</I> oraz \n" +
                "<I>Harry Potter i " +
                  "więzień Azkabanu</I>.",
              25.95)
    };

  public static Item getItem(String itemID) {
    Item 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);
  }
}
```