helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

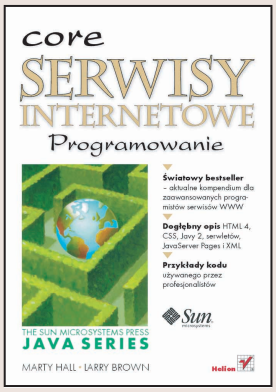
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Serwisy internetowe. Programowanie



Autorzy: Marty Hall, Larry Brown
Tłumaczenie: Szymon Kobalczyk, Marek Pałczyński, Rafał Szpoton
ISBN: 83-7361-155-X
Tytuł oryginału: Core Web Programming, 2nd Edition
Format: B5, stron: 1254
Przykłady na ftp: 1211 kB

Kompendium wiedzy dla profesjonalnych programistów serwisów internetowych.

Książka zawiera komplet informacji niezbędnych twórcom zaawansowanych aplikacji sieciowych i opisuje: HTML 4, Java Standard Edition 1.3, serwlety, JavaServer Pages i wiele innych technologii. „Serwisy internetowe. Programowanie" łączy te technologie pokazując, jak przy ich pomocy tworzyć profesjonalne systemy.

HTML 4: Dogłębny, praktyczny opis struktury dokumentu HTML, jego elementów, ramek i styli kaskadowych.

Java 2: Podstawy składni, projektowanie zorientowane obiektowo, aplety, tworzenie animacji, narzędzie Java PlugIn, tworzenie interfejsu użytkownika (Swing), Java 2D API, aplikacje wielowątkowe, programowanie sieciowe, korzystanie z baz danych i jeszcze więcej.

Technologie serwerowe Javy: Serwlety, JSP, XML, JDBC – fundamentalne technologie umożliwiające budowę aplikacji w Javie. Tematy zaawansowane dotyczą tworzenia własnych bibliotek znaczników JSP, łączenia serwletów z JSP (MVC), tworzenie puli połączeń do baz danych, SAX, DOM i XSLT. Dokładnie omówiono także HTTP 1.1.

JavaScript: Dynamiczne tworzenie zawartości stron WWW, śledzenie zachowania użytkownika, sprawdzanie poprawności danych w formularzach HTML. Dołączono podręczny skorowidz komend.

Pierwsze wydanie tej książki jest używane jako podręcznik na takich uczelniach jak MIT, Standford, UC Berkeley, UCLA, Princeton i John Hopkins. Książka, którą trzymasz w ręku, została zaktualizowana i uzupełniona o najnowsze technologie oraz towarzyszące im przykłady gotowego kodu.

- Światowy bestseller – aktualne kompendium dla zaawansowanych programistów serwisów WWW
- Dogłębny opis HTML 4, CSS, Javy 2, serwletów, JavaServer Pages i XML
- Przykłady kodu używanego przez profesjonalistów

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

Spis treści

O Autorach 21

Wstęp 23

Część I Język HTML 29

Rozdział 1. Projektowanie stron WWW w HTML 4.0 31

1.1. Język HTML 32
1.2. HTML 4.0 oraz inne standardy HTML 33
1.3. Publikacja dokumentu WWW 35
Tworzenie dokumentu 35
Umieszczenie dokumentu w sieci WWW 35
Sprawdzanie poprawność dokumentu 37
1.4. Podstawowa struktura dokumentów HTML 38
Szablon dokumentu HTML 39
Deklaracja DOCTYPE 39
1.5. Nagłówek HEAD — ogólne informacje o stronie 41
Obowiązkowy element nagłówka HEAD 42
Opcjonalne elementy nagłówka HEAD 42
1.6. Sekcja BODY — część główna dokumentu 47
1.7. Podsumowanie 50

Rozdział 2. Elementy blokowe w HTML 4.0 51

2.1. Nagłówki 52
2.2. Podstawowe bloki tekstu 53
Podstawowy akapit 54
Akapit zachowujący znaki rozdzielające 55
Blok cytatu 56
Adresy 56
2.3. Listy numerowane, wypunktowane i wcięte 57
Listy numerowane 57
Listy wypunktowane 59
Listy definicji 61
2.4. Tabele 62
Podstawowa struktura tabel 62
Definiowanie wierszy tabeli 66
Nagłówki kolumn i komórki danych 67
Grupowanie zawartości tabeli 70
2.5. Formularze 72
2.6. Pozostałe elementy blokowe 73
2.7. Podsumowanie 75

Rozdział 3. Elementy tekstowe w HTML 4.0 ... 77
3.1. Fizyczne style znaków ... 77
3.2. Logiczne style znaków ... 82
3.3. Tworzenie łączy hipertekstowych ... 84
3.4. Umieszczanie ilustracji ... 87
Animowane obrazki GIF ... 87
Element IMG ... 88
3.5. Mapy obrazu interpretowane po stronie klienta ... 91
3.6. Osadzenie innych obiektów w dokumentach ... 94
Osadzanie apletów Javy ... 94
Osadzanie plików w formatach audio, wideo i innych przez moduły rozszerzające ... 96
Osadzanie kontrolek ActiveX ... 97
Wyświetlanie ruchomych szyldów ... 99
3.7. Sterowanie łamaniem tekstu ... 100
3.8. Podsumowanie ... 101

Rozdział 4. Ramki ... 103
4.1. Szablon dokumentu ramek ... 104
4.2. Definiowanie układu ramek ... 105
4.3. Definiowanie zawartości komórek ramek ... 109
Przykłady ... 111
4.4. Odwoływanie się do komórek ramek ... 112
Predefiniowane nazwy ramek ... 115
4.5. Rozwiązywanie najczęstszych problemów z ramkami ... 115
Tworzenie zakładek do stron z ramkami ... 116
Drukowanie stron z ramkami ... 116
Uaktualnianie wielu ramek równocześnie ... 117
Chronienie dokumentu przed umieszczeniem w ramce ... 120
Tworzenie pustych komórek ... 120
4.6. Ramki wewnętrzne ... 121
4.7. Podsumowanie ... 124

Rozdział 5. Kaskadowe arkusze stylów ... 125
5.1. Definiowanie reguł stylu ... 126
5.2. Zewnętrzne i lokalne arkusze stylów ... 128
Zewnętrzne arkusze stylów ... 128
Element STYLE i arkusze stylów JavaScript ... 130
Wewnętrzne definicje stylów ... 130
5.2. Wskaźniki ... 131
Elementy HTML ... 132
Klasy definiowane przez użytkownika ... 133
Identyfikatory definiowane przez użytkownika ... 133
Pseudoklasy odsyłaczy ... 134
5.4. Łączenie kaskadowe. Reguły pierwszeństwa w arkuszach stylów ... 135
5.5. Właściwości kroju pisma ... 136
5.6. Właściwości pierwszego planu i tła ... 141
5.7. Właściwości tekstu ... 143
5.8. Właściwości prostokąta ograniczającego ... 147
Marginesy ... 148
Obramowanie ... 148
Dopełnienie ... 150
Tryby wyświetlania prostokąta ograniczającego ... 150

5.9. Obrazki i elementy pływające ... 151
5.10. Właściwości list ... 153
5.11. Standardowe formaty zapisu wartości ... 153
Długości ... 154
Kolory ... 154
5.12. Warstwy ... 155
Definiowanie warstw przez elementy LAYER i ILAYER ... 155
Definiowanie warstw przez arkusze stylów ... 159
5.13. Podsumowanie ... 162

Część II Programowanie w języku Java 163

Rozdział 6. Podstawowe informacje o Javie ... 165

6.1. Cechy charakterystyczne Javy ... 166
Java jest językiem współpracującym z Internetem i aplikacjami sieciowymi ... 167
Java jest językiem przeznaczonym dla różnych platform ... 169
Java jest językiem łatwym do nauczenia ... 172
Java jest językiem obiektowym ... 173
Dzięki standardowym bibliotekom Java ma wiele zastosowań ... 173
6.2. Nieprawdziwe informacje na temat Javy ... 175
Java jest przeznaczona jedynie do zastosowań sieciowych ... 175
Java jest językiem niezależnym od platformy ... 175
Java jest językiem łatwym do nauczenia ... 177
Java jest językiem obiektowym ... 177
Java jest językiem, który można stosować w każdym projekcie informatycznym ... 178
6.3. Wersje Javy ... 178
Której wersji powinno się używać? ... 180
Bez względu na to, która wersja zostanie wybrana… ... 180
6.4. Rozpoczęcie pracy ... 181
Instalacja Javy ... 182
Instalacja przeglądarki obsługującej Javę ... 183
Zapisanie odsyłacza lub instalowanie dokumentacji API Javy ... 183
Opcjonalne instalowanie zintegrowanego środowiska programistycznego ... 184
Utworzenie i uruchomienie programu ... 185
6.5. Przykładowe programy Javy ... 185
Aplikacja „Witaj świecie” ... 186
Argumenty linii poleceń ... 186
Aplet „Witaj świecie” ... 187
Parametry apletu ustalane przez użytkownika ... 188
6.6. Podsumowanie ... 191

Rozdział 7. Programowanie obiektowe w języku Java ... 193

7.1. Zmienne obiektowe ... 194
7.2. Metody ... 196
7.3. Konstruktory i słowo kluczowe „this” ... 197
Inicjacja statyczna ... 200
7.4. Destruktory ... 200
7.5. Przeciążanie ... 201
7.6. Udostępnianie klas w oddzielnych plikach ... 204

7.7. Program javadoc ... 209
Znaczniki javadoc ... 210
Argumenty linii poleceń programu javadoc ... 213
7.8. Dziedziczenie ... 215
7.9. Interfejsy i klasy abstrakcyjne ... 219
7.10. Pakiety, ścieżki dostępu do klas i archiwa JAR ... 226
Ścieżka do klas — CLASSPATH ... 229
7.11. Modyfikatory deklaracji ... 232
Modyfikatory widoczności ... 232
Inne modyfikatory ... 233
7.12. Podsumowanie ... 235

Rozdział 8. Podstawowa składnia języka Java ... 237

8.1. Podstawowe informacje na temat składni ... 238
8.2. Podstawowe typy danych ... 239
Konwersja typów prostych ... 242
8.3. Operatory, instrukcje warunkowe i pętle ... 242
Operatory matematyczne ... 243
Instrukcje warunkowe ... 244
Pętle ... 249
8.4. Klasa Math ... 252
Stałe ... 252
Metody ogólnego przeznaczenia ... 252
Metody trygonometryczne ... 254
Klasy BigInteger i BigDecimal ... 254
8.5. Wprowadzanie i wyświetlanie danych ... 256
Wyprowadzanie danych na standardowe urządzenie wyjściowe ... 256
Wyprowadzanie danych na standardowe wyjście informacji o błędach ... 258
Odczytywanie danych ze standardowego urządzenia wejściowego ... 258
8.6. Uruchamianie programów zewnętrznych ... 258
8.7. Referencje ... 264
Przekazywanie argumentów w Javie ... 266
Operator instanceof ... 266
8.8. Ciągi tekstowe ... 268
Metody klasy String ... 268
Konstruktory ... 274
8.9. Tablice ... 274
Dwuetapowa alokacja tablicy ... 275
Jednoetapowa alokacja tablicy ... 276
Tablice wielowymiarowe ... 277
8.10. Wektory ... 278
Konstruktory ... 278
Metody ... 278
8.11. Przykład — proste drzewo binarne ... 280
8.12. Wyjątki ... 284
Podstawy obsługi wyjątków ... 285
Wielokrotne klauzule catch ... 287
Klauzula Finally ... 288
Generowanie wyjątków ... 288
Niesprawdzane wyjątki ... 290
8.13. Podsumowanie ... 290

Rozdział 9. Aplety i podstawy operacji graficznych ... 291
9.1. Czym są aplety? ... 292
9.2. Budowanie apletu ... 293
Szablony apletów ... 293
Szablony HTML ... 294
9.3. Przykładowy aplet ... 295
Automatyczne odświeżanie wyglądu okna ... 296
Odświeżanie kodu apletu podczas prac projektowych ... 296
9.4. Poszczególne etapy pracy apletu ... 298
9.5. Inne metody apletów ... 299
9.6. Znacznik HTML APPLET ... 304
9.7. Odczyt parametrów apletu ... 306
Przykład odczytu parametrów apletu ... 307
9.8. Znacznik HTML OBJECT ... 309
9.9. Rozszerzenie Javy ... 311
9.10. Aplikacje graficzne ... 313
9.11. Operacje graficzne ... 314
Rysowanie ... 315
Kolory i czcionki ... 317
Tryby kreślenia ... 318
Współrzędne i obszary obcinania ... 319
9.12. Wyświetlanie plików graficznych ... 319
Pobieranie rysunków określonych względnym adresem URL ... 320
Pobieranie rysunków określonych bezwzględnym adresem URL ... 322
Pobieranie rysunków w aplikacjach Javy ... 323
9.13. Buforowanie plików graficznych ... 325
9.14. Nadzorowanie procesu pobierania rysunków ... 328
MediaTracker ... 330
9.15. Podsumowanie ... 334

Rozdział 10. Java 2D — grafika w Java 2 ... 335
10.1. Podstawowe informacje o bibliotece Java 2D ... 336
Najczęściej wykorzystywane metody klasy Graphics2D ... 339
10.2. Style kreślenia ... 342
Klasy kształtu ... 342
10.3. Style wypełnienia ... 346
Klasy wypełniania ... 346
Rysunki jako wzorce wypełnień ... 349
10.4. Przezroczystość ... 353
10.5. Korzystanie z czcionek lokalnych ... 355
10.6. Style kreślenia linii ... 357
Atrybuty pióra ... 357
10.7. Transformacje współrzędnych ... 362
Pochylenie ... 365
10.8. Pozostałe funkcje biblioteki Java 2D ... 366
10.9. Podsumowanie ... 367

Rozdział 11. Obsługa zdarzeń myszy i klawiatury ... 369
11.1. Niezależne klasy nasłuchu zdarzeń ... 370
Rysowanie kół ... 372
11.2. Implementacja interfejsów nasłuchu ... 373

11.3. Obsługa zdarzeń w klasach wewnętrznych ... 374
11.4. Obsługa zdarzeń w anonimowych klasach wewnętrznych ... 376
11.5. Standardowe interfejsy nasłuchu ... 377
11.6. Za kulisami — niskopoziomowe przetwarzanie zdarzeń ... 382
11.7. Pole tekstowe poprawiające błędy ... 385
11.8. Klasa deski kreślarskiej ... 387
Ulepszona deska kreślarska ... 389
11.9. Podsumowanie ... 390

Rozdział 12. Rozmieszczanie komponentów ... 391

12.1. Menedżer rozmieszczenia FlowLayout ... 392
Konstruktory menedżera FlowLayout ... 393
Inne metody menedżera FlowLayout ... 393
12.2. Menedżer rozmieszczenia BorderLayout ... 394
Konstruktory menedżera BorderLayout ... 396
Inne metody menedżera BorderLayout ... 396
12.3. Menedżer rozmieszczenia GridLayout ... 397
Konstruktory menedżera GridLayout ... 397
Inne metody menedżera GridLayout ... 399
12.4. Menedżer rozmieszczenia CardLayout ... 399
Konstruktory menedżera CardLayout ... 402
Inne metody menedżera CardLayout ... 403
12.5. Menedżer rozmieszczenia GridBagLayout ... 403
Obiekt GridBagConstraints ... 404
Przykład ... 406
Konstruktor menedżera GridBagLayout ... 409
Inne metody menedżera GridBagLayout ... 409
12.6. Menedżer rozmieszczenia BoxLayout ... 410
Konstruktor menedżera BoxLayout ... 413
Inne metody menedżera BoxLayout ... 413
12.7. Wyłączenie menedżera rozmieszczenia ... 414
12.8. Efektywne wykorzystanie menedżerów rozmieszczenia ... 415
Wykorzystanie zagnieżdżanych kontenerów ... 416
Wyłączenie menedżera rozmieszczenia w pracy z niektórymi kontenerami ... 419
Operowanie wolną przestrzenią wokół komponentów ... 420
12.9. Podsumowanie ... 423

Rozdział 13. Komponenty AWT ... 425

13.1. Klasa Canvas ... 426
Powoływanie i korzystanie z obiektu Canvas ... 427
Przykład — komponent Kolo ... 428
13.2. Klasa Component ... 430
13.3. Lekkie komponenty języka Java 1.1 ... 436
13.4. Klasa Panel ... 438
Domyślny menedżer rozmieszczenia — FlowLayout ... 438
Tworzenie i posługiwanie się obiektem Panel ... 439
Przykład — zastosowanie obiektu Panel do grupowania komponentów ... 439
13.5. Klasa Container ... 441
13.6. Klasa Applet ... 443
13.7. Klasa ScrollPane ... 443
Tworzenie i posługiwanie się obiektem ScrollPane ... 443
Przykład — obiekt ScrollPane zawierający panel o stu przyciskach ... 443

13.8. Klasa Frame ... 444
Domyślny menedżer rozmieszczenia — BorderLayout ... 445
Korzystanie z obiektu Frame ... 445
Przykłady wykorzystania ramek ... 446
Zamykalna ramka ... 447
Menu ... 448
Inne przydatne metody klasy Frame ... 450
13.9. Serializowanie okien ... 451
Zapis okna na dysku ... 451
Odczyt okna z dysku ... 452
Przykład — zapisywana ramka ... 452
13.10. Klasa Dialog ... 455
Tworzenie i posługiwanie się obiektami Dialog ... 455
Przykład — potwierdzenie zamknięcia okna ... 455
13.11. Klasa FileDialog ... 457
Przykład — wyświetlanie zawartości plików w polu tekstowym ... 457
13.12. Klasa Window ... 459
Domyślny menedżer rozmieszczenia — BorderLayout ... 460
Tworzenie i posługiwanie się obiektem Window ... 460
13.13. Obsługa zdarzeń kontrolek GUI ... 461
Rozproszone przetwarzanie zdarzeń ... 462
Scentralizowane przetwarzanie zdarzeń ... 463
13.14. Klasa Button ... 465
Konstruktory ... 465
Przykład — aplet z trzema przyciskami ... 465
Inne metody klasy Button ... 466
Obsługa zdarzeń przycisków ... 467
13.15. Klasa Checkbox ... 469
Konstruktory ... 470
Przykład — zaznaczone pola wyboru ... 470
Inne metody klasy Checkbox ... 471
Obsługa zdarzeń obiektu Checkbox ... 472
13.16. Grupy pól wyboru (przyciski opcji) ... 472
Konstruktory ... 473
Przykład — pola wyboru a przyciski opcji ... 473
Inne metody klas CheckboxGroup i Checkbox ... 474
Obsługa zdarzeń obiektu CheckboxGroup ... 474
13.17 Klasa Choice ... 475
Konstruktor ... 475
Przykład zastosowania list rozwijanych ... 475
Inne metody klasy Choice ... 476
Obsługa zdarzeń obiektu Choice ... 477
13.18. Klasa List ... 478
Konstruktory ... 479
Przykład — listy pojedynczego i wielokrotnego wyboru ... 479
Inne metody klasy List ... 480
Obsługa zdarzeń obiektu List ... 482
13.19. Klasa TextField ... 486
Konstruktory ... 487
Przykład — tworzenie obiektów TextField ... 487
Inne metody klasy TextField ... 488
Obsługa zdarzeń obiektu TextField ... 490

13.20. Klasa TextArea ... 491
Konstruktory ... 491
Przykład — puste i wypełnione obszary tekstowe ... 492
Inne metody klasy TextArea ... 492
Obsługa zdarzeń obiektu TextArea ... 493
13.21. Klasa Label ... 493
Konstruktory ... 494
Przykład — cztery różne etykiety ... 494
Inne metody klasy Label ... 495
Obsługa zdarzeń obiektu Label ... 496
13.22. Paski przewijania i suwaki ... 497
Konstruktory ... 497
Przykład — różnorodne paski kontrolki suwaków ... 498
Inne metody klasy Scrollbar ... 499
Obsługa zdarzeń obiektu Scrollbar ... 501
13.23. Menu kontekstowe ... 503
Konstruktory ... 503
Przykład — menu kontekstowe apletu ... 503
Inne metody klasy PopupMenu ... 505
Obsługa zdarzeń obiektu PopupMenu ... 505
13.24. Podsumowanie ... 506

Rozdział 14. Podstawy Swing ... 507

14.1. Podstawy Swing ... 508
Różnice pomiędzy technologiami Swing i AWT ... 508
14.2. Komponent JApplet ... 515
14.3. Komponent JFrame ... 517
14.4. Komponent JLabel ... 518
Nowe funkcje — wyświetlanie grafiki, obramowania i treści HTML ... 519
Konstruktory klasy JLabel ... 520
Najczęściej wykorzystywane metody klasy JLabel ... 520
14.5. Komponent JButton ... 522
Nowe elementy — ikony, formatowanie, mnemonika ... 523
Kod HTML w opisie przycisku ... 524
Konstruktory klasy JButton ... 524
Najczęściej wykorzystywane metody klasy JButton (AbstractButton) ... 524
14.6. Komponent JPanel ... 526
Konstruktory klasy JPanel ... 527
Nowe elementy — obramowania ... 527
Najczęściej wykorzystywane metody klasy BorderFactory ... 528
14.7. Komponent JSlider ... 530
Nowe elementy — znaczniki przesunięcia i opisy ... 531
Konstruktory klasy JSlider ... 531
Najczęściej wykorzystywane metody klasy JSlider ... 532
14.8. Komponent JColorChooser ... 534
Konstruktory klasy JColorChooser ... 535
Najczęściej wykorzystywane metody klasy JColorChooser ... 535
14.9. Komponent JInternalFrame ... 537
Konstruktory klasy JInternalFrame ... 538
Najczęściej wykorzystywane metody klasy JInternalFrame ... 538
14.10. Komponent JOptionPane ... 541
Najczęściej wykorzystywane metody klasy JOptionPane ... 541

14.11. Komponent JToolBar ... 545
Konstruktory klasy JToolBar ... 546
Najczęściej wykorzystywane metody klasy JToolBar ... 546
14.12 Komponent JEditorPane ... 550
Podążanie za odsyłaczami ... 551
Konstruktory klasy JEditorPane ... 552
Najczęściej wykorzystywane metody klasy JEditorPane ... 552
Implementacja prostej przeglądarki internetowej ... 553
Obsługa kodu HTML i pakiet JavaHelp ... 556
14.13. Inne komponenty Swing ... 556
Komponent JCheckBox ... 557
Komponent JRadioButton ... 558
Komponent JTextField ... 559
Komponent JTextArea ... 560
Komponent JFileChooser ... 560
14.14. Podsumowanie ... 561

Rozdział 15. Swing dla zaawansowanych ... 563

15.1. Wykorzystanie niestandardowych modeli danych i wyświetlaczy ... 564
15.2. JList ... 565
Kontrolka JList ze stałym zestawem opcji ... 565
Kontrolka JList ze zmiennym zestawem opcji ... 570
Kontrolka JList z niestandardowym modelem danych ... 572
Kontrolka JList z niestandardowym wyświetlaczem ... 577
15.3. JTree ... 581
Proste drzewo JTree ... 581
Obsługa zdarzeń JTree ... 585
Niestandardowe modele danych i dynamiczne drzewa ... 587
Zastępowanie ikon dla wierzchołków drzewa ... 590
15.4. JTable ... 592
Prosta tabela JTable ... 592
Modele danych tabeli ... 596
Wyświetlacze komórek tabeli ... 601
Obsługa zdarzeń tabeli ... 603
15.5. Drukowanie komponentów Swing ... 606
Podstawowe zasady drukowania ... 607
Rola podwójnego buforowania ... 609
Ogólna procedura drukowania komponentów ... 609
Drukowanie w JDK 1.3 ... 614
15.6. Wielowątkowość w Swing ... 615
Metody klasy SwingUtilities ... 618
15.7. Podsumowanie ... 620

Rozdział 16. Programowanie równoległe z wykorzystaniem wielowątkowości ... 621

16.1. Uruchamianie wątków ... 622
Metoda 1 — umieszczenie kodu w odrębnym obiekcie wątku ... 623
Metoda 2 — umieszczenie kodu w klasie sterującej, implementującej interfejs Runnable ... 625
16.2. Sytuacje wyścigu ... 627
16.3. Synchronizacja ... 630
Synchronizacja sekcji kodu ... 630
Synchronizacja całej metody ... 631
Powszechny błąd przy synchronizacji ... 631

16.4. Tworzenie metod wielowątkowych ... 633
16.5. Metody klasy Thread ... 638
Konstruktory ... 638
Stałe ... 639
Metody ... 640
Zatrzymywanie wątku ... 644
16.6. Grupy wątków ... 646
Konstruktory ... 646
Metody ... 647
16.7. Wielowątkowa grafika i podwójne buforowanie ... 648
Odrysowywanie wszystkiego w metodzie paint ... 649
Zaimplementowanie elementów dynamicznych jako odrębnego komponentu ... 652
Stworzenie odrębnych procedur bezpośrednio zajmujących się rysowaniem ... 653
Nadpisanie metody update i spowodowanie, aby paint wykonywało przyrostowe uaktualnianie ... 655
Wykorzystanie podwójnego buforowania ... 659
16.8. Animowanie obrazków ... 664
16.9. Czasomierze ... 668
Konstruktor ... 671
Inne metody ... 671
16.10. Podsumowanie ... 673

Rozdział 17. Programowanie aplikacji sieciowych ... 675

17.1. Implementacja klienta ... 676
Przykład — ogólny klient sieciowy ... 679
Rezultat — połączenie z serwerem FTP ... 681
17.2. Analiza składniowa łańcuchów za pomocą klasy StringTokenizer ... 682
Klasa StringTokenizer ... 682
Konstruktory ... 682
Metody ... 683
Przykład — interaktywny analizator składniowy ... 683
17.3. Przykład — klient sprawdzający poprawność adresów poczty elektronicznej ... 684
17.4. Przykład — klient sieciowy pobierający dokumenty WWW ... 687
Klasa pobierająca dokument wskazany przez adres URI na podanym serwerze ... 688
Klasa pobierająca dokument wskazany przez adres URL ... 689
Rezultat wykonania programu CzytnikUrl ... 690
17.5. Klasa URL ... 691
Pobieranie dokumentów za pomocą klasy URL ... 691
Inne przydatne metody klasy URL ... 693
17.6. Klient WWW — interaktywna komunikacja z serwerem WWW ... 694
17.7. Implementacja serwera ... 701
Przykład — ogólny serwer sieciowy ... 703
Rezultat — obsługa połączenia z przeglądarki internetowej ... 706
Połączenie pomiędzy programami KlientSieciowy i SerwerSieciowy ... 706
17.8. Przykład — prosty serwer HTTP ... 707
Program WielowatkowySerwerEcho ... 710
17.9. RMI — zdalne wywoływanie metod ... 712
Etapy tworzenia aplikacji RMI ... 713
Prosty przykład ... 714
Praktyczny przykład — serwer wykonujący całkowanie numeryczne ... 718
Praktyczny przykład czterech wymaganych klas ... 720

Kompilacja i uruchomienie przykładu ... 724
Konfiguracja RMI w dużych sieciach ... 725
Kompilacja i uruchomienie przykładu ... 728
Przykład apletu RMI ... 730
17.10. Podsumowanie ... 732

Cześć III Programowanie aplikacji działających po stronie serwera 735

Rozdział 18. Formularze w języku HTML ... 737

18.1. Sposób przesyłania danych w języku HTML ... 738
18.2. Znacznik FORM ... 741
18.3. Elementy tekstowe ... 745
Pola edycyjne ... 746
Pola edycyjne do wprowadzania haseł ... 747
Obszary tekstowe ... 748
18.4. Przyciski ... 750
Przyciski typu Submit ... 750
Przyciski typu Reset ... 753
Przyciski wykorzystywane przez kod JavaScript ... 754
18.5. Pola wyboru oraz przełączniki ... 755
Pola wyboru ... 755
Przełączniki ... 756
18.6 Listy rozwijane oraz zwykłe ... 757
18.7. Wysyłanie plików do serwera ... 760
18.8. Mapy obrazowe po stronie serwera ... 762
IMAGE — standardowe mapy obrazowe po stronie serwera ... 762
ISMAP — alternatywne mapy obrazowe po stronie serwera ... 764
18.9. Pola ukryte ... 766
18.10. Elementy grupujące ... 766
18.11. Kolejność tabulacji ... 768
18.12. Podsumowanie ... 769

Rozdział 19. Programy w języku Java po stronie serwera — serwlety ... 771

19.1. Przewaga serwerów nad tradycyjnymi programami CGI ... 773
Wydajność ... 773
Wygoda ... 773
Duże możliwości ... 773
Przenośność ... 774
Bezpieczeństwo ... 774
Niski koszt ... 774
19.2. Instalacja oraz konfiguracja serwera ... 775
Pobieranie oprogramowania obsługującego serwlety oraz JSP ... 775
Dokumentacja API serwletów oraz stron JSP ... 776
Określanie klas wykorzystywanych przez kompilator Javy ... 777
Umieszczanie klas w pakietach ... 777
Konfiguracja serwera ... 777
Kompilacja oraz instalacja serwletów ... 778
Wywoływanie serwletów ... 779
19.3. Podstawowa struktura serwletu ... 779
Serwlet tworzący prosty tekst ... 780
Serwlet tworzący kod HTML ... 781
Proste narzędzia do tworzenia dokumentów HTML ... 783

19.4. Czas życia serwletu ... 785
Metoda init ... 785
Metoda service ... 786
Metody doGet, doPost oraz doXxx ... 787
Interfejs SingleThreadModel ... 787
Metoda destroy ... 788
19.5. Przykład wykorzystania parametrów inicjujących ... 788
19.6. Żądanie klienta — dostęp do danych formularza ... 791
Odczytywanie danych formularza w programach CGI ... 791
Odczytywanie danych formularza w serwletach ... 791
Przykład — odczytywanie wartości trzech parametrów ... 792
Filtrowanie danych zapytania ... 794
19.7. Żądanie klienta — nagłówki żądania HTTP ... 796
Odczytywanie nagłówków żądania w serwletach ... 796
Przykład — tworzenie tabeli zawierającej wszystkie nagłówki żądania ... 798
Nagłówki żądania protokołu HTTP 1.1 ... 800
Wysyłanie skompresowanych stron WWW ... 803
19.8. Odpowiedniki standardowych zmiennych CGI w przypadku serwletów ... 805
19.9. Odpowiedź serwera — kody stanu HTTP ... 808
Określanie kodów stanu ... 808
Kody stanu protokołu HTTP 1.1. ... 810
Interfejs użytkownika dla różnych wyszukiwarek internetowych ... 815
19.10. Odpowiedź serwera — nagłówki odpowiedzi HTTP ... 819
Ustawianie nagłówków odpowiedzi w serwletach ... 819
Nagłówki odpowiedzi protokołu HTTP 1.1 ... 820
Trwały stan serwletu oraz automatyczne odświeżanie stron ... 826
19.11. Cookies ... 834
Korzyści wynikające ze stosowania cookies ... 835
Niektóre problemy z plikami cookies ... 836
Interfejs programistyczny API obsługi plików Cookie w serwletach ... 837
Przykład ustawiania oraz odczytywania cookies ... 840
Proste narzędzia do obsługi cookies ... 844
Odszukiwanie cookies o określonych nazwach ... 844
Tworzenie długotrwałych cookies ... 845
19.12. Śledzenie sesji ... 846
Konieczność śledzenia sesji ... 846
Interfejs API służący do śledzenia sesji ... 848
Kończenie sesji ... 851
Serwlet ukazujący licznik odwiedzin ... 852
19.13. Podsumowanie ... 854

Rozdział 20. Technologia JavaServer Pages ... 855

20.1. Podstawy technologii JSP ... 855
20.2. Korzyści wynikające ze stosowania technologii JSP ... 857
Porównanie z technologią Active Server Pages (ASP) oraz ColdFusion ... 857
Porównanie z technologią PHP ... 857
Porównanie z serwletami ... 858
Porównanie z technologią Server-Side Includes (SSI) ... 858
Porównanie z językiem JavaScript ... 858
20.3. Elementy skryptowe JSP ... 859
Wyrażenia ... 859
Skryptlety ... 863

Deklaracje ... 865
Zmienne zdefiniowane ... 867
20.4. Dyrektywa JSP o nazwie page ... 868
Atrybut import ... 869
Atrybut contentType ... 871
Atrybut isThreadSafe ... 872
Atrybut session ... 873
Atrybut buffer ... 874
Atrybut autoflush ... 874
Atrybut extends ... 874
Atrybut info ... 874
Atrybut errorPage ... 875
Atrybut isErrorPage ... 875
Atrybut language ... 875
Składnia dyrektyw w postaci zgodnej z XML ... 875
20.5. Dołączanie apletów oraz plików w dokumentach JSP ... 876
Dyrektywa include — dołączanie plików w czasie przekształcania strony do postaci serwletu ... 876
Dołączanie plików w trakcie obsługi żądania ... 879
Dołączanie apletów ... 880
Znacznik jsp:fallback ... 884
20.6. Wykorzystywanie w stronach JSP komponentów JavaBeans ... 888
Prosty przykład wykorzystania klasy ziarna ... 889
Prosty przykład ustawiania właściwości ziarna ... 891
Modyfikacja właściwości ziarna ... 893
Współdzielenie ziaren ... 899
20.7. Definiowanie własnych znaczników JSP ... 902
Komponenty tworzące bibliotekę znaczników ... 903
Definiowanie prostego znacznika ... 906
Przyporządkowanie znacznikom atrybutów ... 910
Atrybuty znaczników — plik deskryptora biblioteki znaczników ... 911
Dołączanie zawartości znacznika ... 913
Opcjonalne dołączanie zawartości znacznika ... 917
Modyfikacja zawartości znacznika ... 920
Znaczniki zagnieżdżone ... 926
20.8. Integrowanie serwletów ze stronami JSP ... 932
Przekazywanie żądań ... 934
Przykład — internetowe biuro podróży ... 937
Przekazywanie żądań ze stron JSP ... 943
20.9. Podsumowanie ... 944

Rozdział 21. Wykorzystywanie apletów w charakterze interfejsu programów działających po stronie serwera ... 945

21.1. Wysyłanie danych przy użyciu metody GET i wyświetlanie wynikowej strony ... 946
21.2. Interfejs użytkownika programu wyszukującego informacje w wielu różnych wyszukiwarkach internetowych ... 947
21.3. Wykorzystywanie metody GET oraz przetwarzanie wyników w sposób bezpośredni (tunelowanie HTTP) ... 951
Odczytywanie danych binarnych oraz danych tekstowych ... 952
Odczytywanie struktur danych zachowanych przy użyciu serializacji ... 953
21.4. Przeglądarka żądań wysyłanych do serwera, używająca serializacji obiektów oraz tunelowania HTTP ... 955

21.5. Wysyłanie danych przy użyciu metody POST
i przetwarzanie wyników wewnątrz apletu (tunelowanie HTTP) ... 962
21.6. Aplet wysyłający dane przy użyciu metody POST ... 965
21.7. Pomijanie serwera HTTP podczas komunikacji z programem po stronie serwera ... 969
21.8. Podsumowanie ... 969

Rozdział 22. Technologia JDBC ... 971

22.1. Podstawowe czynności konieczne do wykonania w celu zastosowania JDBC ... 972
Załadowanie sterownika JDBC ... 972
Określanie adresu URL służącego do nawiązania połączenia ... 973
Ustanawianie połączenia z bazą danych ... 974
Tworzenie instrukcji SQL ... 974
Wykonywanie zapytania ... 975
Przetwarzanie wyników zapytania ... 975
Zamykanie połączenia ... 976
22.2. Prosty przykład wykorzystujący JDBC ... 976
22.3. Kilka narzędzi JDBC ... 981
22.4. Stosowanie utworzonych narzędzi JDBC ... 989
22.5. Interaktywna przeglądarka zapytań ... 993
Kod przeglądarki zapytań ... 995
22.6. Zapytania prekompilowane ... 999
22.7. Podsumowanie ... 1003

Rozdział 23. Przetwarzanie dokumentów XML w języku Java ... 1005

23.1. Analiza dokumentów XML przy użyciu modelu DOM (poziom 2) ... 1006
Instalacja oraz konfiguracja ... 1007
Analiza dokumentu ... 1008
23.2. Przykład wykorzystania modelu DOM — wyświetlenie dokumentu XML
w postaci obiektu JTree ... 1009
23.3. Przetwarzanie dokumentów XML za pomocą interfejsu SAX 2.0 ... 1020
Instalacja oraz konfiguracja ... 1020
Analiza dokumentu ... 1021
23.4. Pierwszy przykład użycia interfejsu SAX — wyświetlanie struktury dokumentu XML ... 1023
23.5. Drugi przykład użycia interfejsu SAX — zliczanie zamówień książek ... 1029
23.6. Przekształcanie dokumentu XML za pomocą transformacji XSLT ... 1033
Instalacja oraz konfiguracja ... 1034
Transformacja ... 1035
23.7. Pierwszy przykład XSLT — edytor dokumentów XSLT ... 1039
23.8. Drugi przykład XSLT — własny zdefiniowany znacznik JSP ... 1047
23.9. Podsumowanie ... 1054

Część IV JavaScript 1055

Rozdział 24. JavaScript — wzbogacanie stron WWW o elementy dynamiczne ... 1057

24.1. Dynamiczne generowanie kodu HTML ... 1059
Zachowanie zgodności z wieloma przeglądarkami ... 1062
24.2. Monitorowanie zdarzeń użytkownika ... 1064
24.3. Poznawanie składni JavaScript ... 1065
Dynamiczne określanie typu ... 1065
Deklarowanie funkcji ... 1066
Klasy i obiekty ... 1067
Tablice ... 1071

24.4. Wykorzystanie JavaScript do przekształcania stron WWW ... 1072
Dostosowanie do rozmiaru okna przeglądarki ... 1073
Sprawdzanie, jakie moduły rozszerzające są dostępne ... 1075
24.5. Wykorzystanie JavaScript do umieszczenia na stronie elementów dynamicznych ... 1077
Dynamiczna zmiana obrazków ... 1077
Przemieszczanie warstw ... 1084
24.6. Wykorzystanie JavaScript do sprawdzania poprawności formularzy ... 1087
Sprawdzanie pojedynczych wartości ... 1088
Sprawdzanie wartości przed odesłaniem formularza ... 1090
24.7. Wykorzystanie JavaScript do zapisywania i sprawdzania znaczników kontekstu klienta ... 1094
24.8. Wykorzystanie JavaScript do współdziałania z ramkami ... 1098
Pokierowanie ramką, aby wyświetliła podany URL ... 1099
Sprawianie aby ramka była aktywna ... 1102
24.9. Wywoływanie Javy z poziomu JavaScript ... 1102
Bezpośrednie wywoływanie metod Javy ... 1103
Wykorzystanie apletów do wykonywania operacji dla JavaScript ... 1104
Sterowanie apletami przy użyciu JavaScript ... 1107
24.10. Dostęp do obiektów JavaScript z poziomu Javy ... 1110
Przykład — dopasowanie koloru tła apletu do koloru strony ... 1112
Przykład — aplet sterujący wartościami formularza HTML ... 1113
Metody klasy JSObject ... 1122
24.11. Podsumowanie ... 1123

Rozdział 25. Krótki podręcznik JavaScript ... 1125

25.1. Obiekt Array ... 1125
Konstruktory ... 1126
Właściwości ... 1126
Metody ... 1126
Zdarzenia ... 1129
25.2. Obiekt Button ... 1129
Właściwości ... 1129
Metody ... 1129
Zdarzenia ... 1130
25.3. Obiekt Checkbox ... 1130
Właściwości ... 1131
Metody ... 1131
Zdarzenia ... 1131
25.4. Obiekt Date ... 1132
Konstruktory ... 1132
Właściwości ... 1132
Metody ... 1133
Zdarzenia ... 1135
25.5. Obiekt Document ... 1135
Właściwości ... 1135
Metody ... 1137
Zdarzenia ... 1138
25.6. Obiekt Element ... 1138
Właściwości ... 1138
Metody ... 1139
Zdarzenia ... 1140

25.7. Obiekt FileUpload ... 1140
Właściwości ... 1140
Metody ... 1141
Zdarzenia ... 1141
25.8. Obiekt Form ... 1141
Właściwości ... 1141
Metody ... 1142
Zdarzenia ... 1142
25.9. Obiekt Function ... 1143
Konstruktor ... 1143
Właściwości ... 1143
Metody ... 1144
Zdarzenia ... 1144
25.10. Obiekt Hidden ... 1144
Właściwości ... 1144
Metody ... 1144
Zdarzenia ... 1145
25.11. Obiekt History ... 1145
Właściwości ... 1145
Metody ... 1145
Zdarzenia ... 1146
25.12. Obiekt Image ... 1146
Konstruktor ... 1146
Właściwości ... 1146
Metody ... 1147
Zdarzenia ... 1147
25.13. Obiekt JavaObject ... 1148
25.14. Obiekt JavaPackage ... 1148
25.15. Obiekt Layer ... 1148
Konstruktory ... 1149
Właściwości ... 1149
Metody ... 1150
Zdarzenia ... 1151
25.16. Obiekt Link ... 1152
Właściwości ... 1152
Metody ... 1153
Zdarzenia ... 1153
25.17. Obiekt Location ... 1154
Właściwości ... 1154
Metody ... 1155
Zdarzenia ... 1155
25.18. Obiekt Math ... 1155
Właściwości ... 1155
Metody ... 1156
Zdarzenia ... 1157
25.19. Obiekt MimeType ... 1158
Właściwości ... 1158
Metody ... 1158
Zdarzenia ... 1158

25.20. Obiekt Navigator ... 1159
Właściwości ... 1159
Metody ... 1161
Zdarzenia ... 1161
25.21. Obiekt Number ... 1161
Konstruktor ... 1162
Właściwości ... 1162
Metody ... 1162
Zdarzenia ... 1164
25.22. Obiekt Object ... 1164
Konstruktory ... 1164
Właściwości ... 1164
Metody ... 1165
Zdarzenia ... 1165
25.23. Obiekt Option ... 1165
Konstruktory ... 1165
Właściwości ... 1166
Metody ... 1166
Zdarzenia ... 1166
25.24. Obiekt Password ... 1166
Właściwości ... 1167
Metody ... 1167
Zdarzenia ... 1167
25.25. Obiekt Plugin ... 1168
Właściwości ... 1168
Metody ... 1169
Zdarzenia ... 1169
25.26. Obiekt Radio ... 1169
Właściwości ... 1169
Metody ... 1170
Zdarzenia ... 1170
25.27. Obiekt RegExp ... 1170
Konstruktory ... 1170
Właściwości ... 1172
Metody ... 1173
Zdarzenia ... 1174
Wzorce specjalne w wyrażeniach regularnych ... 1174
25.28. Obiekt Reset ... 1174
Właściwości ... 1174
Metody ... 1174
Zdarzenia ... 1176
25.29. Obiekt Screen ... 1176
Właściwości ... 1177
Metody ... 1177
Zdarzenia ... 1177
25.30. Obiekt Select ... 1177
Właściwości ... 1178
Metody ... 1179
Zdarzenia ... 1179

25.31. Obiekt String ... 1179
Konstruktor ... 1180
Właściwości ... 1180
Metody ... 1180
Zdarzenia ... 1184
25.32. Obiekt Submit ... 1184
Właściwości ... 1184
Metody ... 1185
Zdarzenia ... 1185
25.33. Obiekt Text ... 1186
Właściwości ... 1186
Metody ... 1186
Zdarzenia ... 1187
25.34. Obiekt Textarea ... 1187
Właściwości ... 1187
Metody ... 1188
Zdarzenia ... 1188
25.35. Obiekt Window ... 1189
Właściwości ... 1189
Metody ... 1192
Zdarzenia ... 1196
Przykład użycia metody open ... 1198
25.36. Podsumowanie ... 1200

Dodatki **1201**

Skorowidz ... 1203

21

Wykorzystywanie apletów w charakterze interfejsu programów działających po stronie serwera

W tym rozdziale:

- Wysyłanie danych przy użyciu metody GET i wyświetlanie wyników w przeglądarce
- Wysyłanie danych przy użyciu metody GET i przetwarzanie wyników wewnątrz apletu (tunelowanie HTTP)
- Wykorzystanie serializacji obiektów do wymiany struktur danych wysokiego poziomu pomiędzy apletami a serwletami
- Wysyłanie danych przy użyciu metody POST i przetwarzanie wyników wewnątrz apletu (tunelowanie HTTP)
- Pomijanie serwera HTTP podczas komunikacji z programem po stronie serwera

Omówione w rozdziale 18. formularze HTTP stanowią prosty, lecz ograniczony sposób pobierania danych wejściowych od użytkownika i przekazywania ich do serwletów lub programów CGI. Niekiedy jednak wymagane jest zastosowanie bardziej skomplikowanego interfejsu użytkownika. Aplety umożliwiają większą kontrolę na rozmiarem, kolorem oraz czcionką elementów graficznego interfejsu użytkownika, jak również udostępniają więcej możliwości obsługiwanych przez dostarczone obiekty (paski postępu, przewijania, rysowanie linii, okna modalne itp.), pozwalają śledzić zdarzenia myszy oraz klawiatury, obsługiwać definiowane własne formularze (przeciągane ikony, klawiatury do wybierania numerów itp.), a także pozwalają wysłać żądanie pochodzące od jednego użytkownika do wielu programów działających po stronie serwera. Te dodatkowe możliwości mają jednak swoją

cenę, ponieważ stworzenie interfejsu użytkownika w języku Java pociąga za sobą więcej wysiłku, niż ma to miejsce w przypadku formularzy HTML, szczególnie wtedy, gdy interfejs zawiera tekst formatowany. Dlatego też wybór pomiędzy formularzami HTML a apletami zależy od rodzaju tworzonej aplikacji.

W formularzach HTML żądania typu `GET` oraz `POST` obsługiwane są w prawie ten sam sposób. Wszystkie elementy interfejsu użytkownika służące do wprowadzania danych są identyczne. Zmieniana jest jedynie wartość atrybutu `METHOD` znacznika `FORM`. Natomiast w przypadku apletów istnieją trzy różne podejścia do problemu. Pierwsze z nich, opisane w podrozdziale 21.1, polega na naśladowaniu przez aplety formularzy HTML używających metody `GET`, co powoduje wysłanie danych do serwera właśnie przy użyciu tej metody i wyświetlenie wyników w przeglądarce. W podrozdziale 21.2, „Interfejs użytkownika programu wyszukującego informacje w wielu różnych wyszukiwarkach internetowych", przedstawiony zostanie przykład tego rodzaju podejścia. Druga metoda zostanie opisana w podrozdziale 21.3. i polega na wykorzystaniu apletu wysyłającego dane do serwletu, a następnie samodzielnie przetwarzającego uzyskane wyniki. Przykład prezentujący zastosowanie tej metody przedstawiony będzie w podrozdziale 21.4, „Przeglądarka żądań wysyłanych do serwera używająca serializacji obiektów oraz tunelowania HTTP". Trzecie podejście, opisane w podrozdziale 21.5, polega na wykorzystaniu apletu wysyłającego dane do serwletu przy użyciu metody POST, a następnie samodzielnie przetwarzającego uzyskane wyniki. Przykład tej metody zamieszczony jest w podrozdziale 21.6, „Aplet wysyłający dane przy użyciu metody POST". W podrozdziale 21.7. przedstawiony zostanie sposób całkowitego pominięcia przez aplet serwera HTTP i bezpośredniej komunikacji z programem działającym na tym samym komputerze, z którego pobrany został aplet.

W rozdziale tym założono, że Czytelnik w stopniu podstawowym zna sposoby tworzenia podstawowych apletów (rozdział 9.) i dlatego skoncentrowano się jedynie na technikach umożliwiających komunikację apletów z programami działającymi po stronie serwera.

21.1. Wysyłanie danych przy użyciu metody GET i wyświetlanie wynikowej strony

Metoda `showDocument` nakazuje przeglądarce wyświetlenie dokumentu umieszczonego pod określonym adresem URL. Dzięki dołączeniu do adresu URL programu danych, umieszczonych po znaku zapytania (?), możliwe jest przesłanie ich przy użyciu metody `GET` do serwletu lub innego programu CGI. Dzięki temu w celu wysłania danych z apletu konieczne jest jedynie dołączenie ich do łańcucha znakowego, służącego do tworzenia adresu URL, a następnie w normalny sposób utworzenie obiektu URL i wywołanie metody `showDocument`. Poniżej przedstawiony został prosty szablon, pozwalający na zastosowanie tej metody w apletach. Zakłada on, że `bazowyAdresURL` zawiera łańcuch znakowy, reprezentujący adres URL programu po stronie serwera, zaś zmienna `pewneDane` zawiera informacje, które mają zostać wysłane wraz z żądaniem.

```
try {
  URL adresProgramu = new URL(bazowyAdresURL + "?" + pewneDane);
  getAppletContext().showDocument(adresProgramu);
} catch (MalformedURLException mue) { ... }
```

Kiedy dane przesyłane są przez przeglądarkę, zostaną zakodowane przy użyciu metody *URL encode*, co oznacza, że spacje konwertowane są na znaki plusa (+), zaś wszystkie nie alfanumeryczne znaki zmieniane są na znak procenta (%), po którym występują dwie cyfry szesnastkowe, określające kod znaku w używanym zestawie. Zostało to dokładnie opisane w podrozdziale 18.2, „Znacznik FORM". W poprzednim przykładzie założono, że łańcuch znakowy `pewneDane` jest już zakodowany poprawnie. W przeciwnym wypadku wystąpiłby błąd działania programu. W JDK 1.1 oraz w wersjach późniejszych występuje specjalna klasa o nazwie `URLEncoder`, posiadająca zdefiniowaną statyczną metodę o nazwie `encode`, której wywołanie powoduje wykonanie omówionego kodowania. Jeśli więc aplet komunikuje się z programem po stronie serwera, który normalnie otrzymywałby dane `GET` z formularzy HTML, aplet musi zakodować każdą przesyłaną wartość, lecz musi również użyć znaku równości (=) pomiędzy każdą nazwą zmiennej a stowarzyszoną z nią wartością, jak również znaku (&) w celu oddzielenia od siebie każdej takiej pary. Dlatego też nie można wykorzystać wspomnianej metody `URLEncoder.encode(pewneDane)` dla całego łańcucha znakowego zawierającego dane (wówczas zostałyby zakodowane również te znaki specjalne — *przyp. tłum.*), lecz w zamian selektywnie zakodować każdą część pary. Czynność ta może zostać wykonana na przykład w poniższy sposób:

```
String pewneDane =
nazwa1 + "=" + URLEncoder.encode(wartosc1) + "&" +
nazwa2 + "=" + URLEncoder.encode(wartosc2) + "&" +
...
nazwaN + "=" + URLEncoder.encode(wartoscN);
try {
  URL adresProgramu = new URL(bazowyAdresURL + "?" + pewneDane);
  getAppletContext().showDocument(adresProgramu);
} catch (MalformedURLException mue) { ... }
```

Cały przykład zostanie przedstawiony w kolejnym podrozdziale.

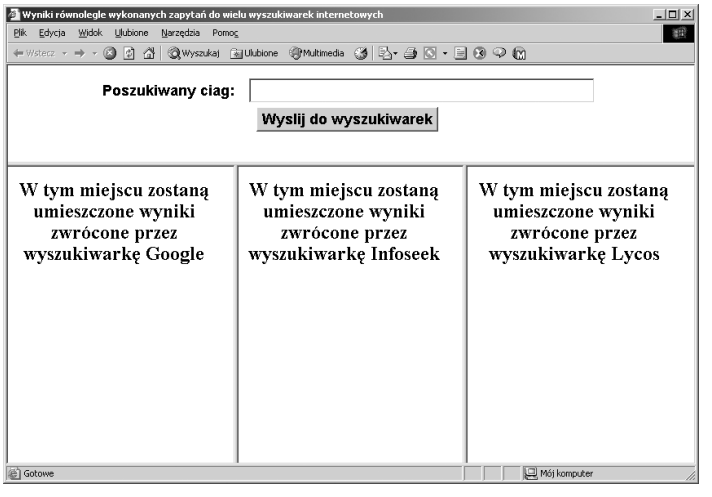
21.2. Interfejs użytkownika programu wyszukującego informacje w wielu różnych wyszukiwarkach internetowych

Na listingu 21.1 przedstawiony został aplet tworzący pole tekstowe, służące do pobierania danych wejściowych od użytkownika. Po zatwierdzeniu danych, aplet koduje je używając metody URL-encode i tworzy trzy różne adresy URL, z dołączanymi do nich danymi przesyłanymi za pomocą metody `GET`. Każdy z nich przeznaczony jest dla innej wyszukiwarki internetowej: Google, Infoseek oraz Lycos. Następnie, w celu nakazania przeglądarce wyświetlenia rezultatów wyszukiwania w trzech różnych obszarach ramki, aplet używa metody `showDocument`. Formularze HTML nie mogą zostać wykorzystane do wykonania tego rodzaju zadania, ponieważ mogą one przesyłać dane jedynie do pojedynczego adresu URL. Na listingu 21.2 przedstawiona została klasa o nazwie `ParametryWyszukiwania`, wykorzystywana przez aplet do tworzenia określonych adresów URL, niezbędnych do przekierowania żądań do różnych wyszukiwarek internetowych. Jak wspomniano w rozdziale 19., klasa `ParametryWyszukiwania` może być również wykorzystywana przez serwlety. Wyniki działania apletu przedstawiają rysunki 21.2 – 21.2.

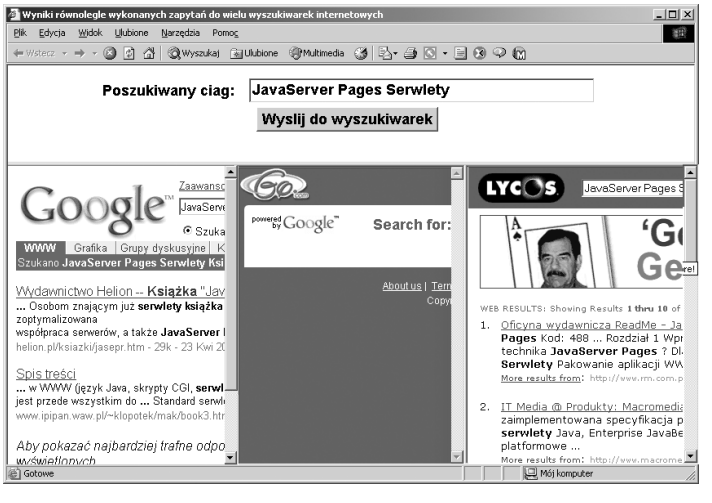
Na listingu 21.3 zaprezentowany został główny dokument HTML, zaś na listingu 21.4 dokument HTML wykorzystywany do osadzenia apletu. Zainteresowanych dokładną budową tych trzech niewielkich plików HTML, służących do utworzenia początkowej zawartości trzech komórek ramki, ukazanych na rysunku 21.1, odsyłamy do kodu źródłowego umieszczonego w archiwum znajdującym się po adresem: *ftp://ftp.helion.pl/przyklady/serinp.zip*.

Listing 21.1. *ApletWyszukiwania.java*

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

/** Aplet odczytujący wartość umieszczoną w polu edycyjnym,
 *  a następnie tworzący przy jej użyciu trzy różne adresy URL
 *  zawierające osadzone dane GET: po jednym dla wyszukiwarek Google, Infoseek oraz Lycos.
 *  Następnie adresy te przekazywane są do przeglądarki
 *  wyświetlającej wyniki wyszukiwania w trzech oddzielnych ramkach.
 *  Zwróć uwagę, że zwykłe formularze HTML nie są w stanie w sposób automatyczny
 *  wysyłać wielu jednoczesnych zapytań.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class ApletWyszukiwania extends Applet
                          implements ActionListener {
  private TextField poleZapytania;
  private Button przyciskZatwierdzajacy;

  public void init() {
    setBackground(Color.white);
    setFont(new Font("Serif", Font.BOLD, 18));
    add(new Label("Poszukiwany ciąg:"));
    poleZapytania = new TextField(40);
    poleZapytania.addActionListener(this);
    add(poleZapytania);
    przyciskZatwierdzajacy = new Button("Wyślij do wyszukiwarek");
    przyciskZatwierdzajacy.addActionListener(this);
    add(przyciskZatwierdzajacy);
  }

  /** Wysyła dane w przypadku wciśnięcia przycisku <B>lub</B>
   *  użycia klawisza Return w polu tekstowym.
   */

  public void actionPerformed(ActionEvent event) {
    String zapytanie = URLEncoder.encode(poleZapytania.getText());
    ParametryWyszukiwania [] glowneAdresy = ParametryWyszukiwania.pobierzGlowneAdresy();
    // Pominięty został adres HotBot (ostatni wpis), ponieważ w celu wyświetlenia wyników
    // wyszukiwarka ta używa kodu w jezyku JavaScript. Dlatego też poniżej występuje
    wyrażenie length-1 .
    for(int i=0; i<glowneAdresy.length-1; i++) {
      try {
```



Rysunek 21.1. *Aplet SearchApplet pozwala użytkownikom na wprowadzenie jednego poszukiwanego ciągu znaków, wykorzystanego następnie w trzech różnych wyszukiwarkach internetowych*



Rysunek 21.2. *Zatwierdzenie zapytania powoduje wyświetlenie obok siebie wyników wyszukiwania, pochodzących z trzech różnych wyszukiwarek*

```
            ParametryWyszukiwania adres = glowneAdresy[i];
            // Klasa SearchSpec tworzy adresy URL
            // w postaci wymaganej przez kilka różnych powszechnie używanych wyszukwiarek
            internetowych.
            URL searchURL = new URL(adres.utworzAdresURL(zapytanie, "10"));
            String nazwaRamki = "results" + i;
            getAppletContext().showDocument(searchURL, nazwaRamki);
          } catch(MalformedURLException mue) {}
        }
      }
    }
```

Listing 21.2. *ParametryWyszukiwania.java*

```
/** mała klasa określająca sposób tworzenia poszukiwanego ciągu
 *  wykorzystywanego w przypadku określonej wyszukiwarki.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
```

```
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class ParametryWyszukiwania {
  private String nazwa, adresURL, znacznikLiczbyWynikow;

  private static ParametryWyszukiwania[] glowneAdresy =
    { new ParametryWyszukiwania("google",
                     "http://www.google.com/search?q=",
                     "&num="),
      new ParametryWyszukiwania("infoseek",
                     "http://infoseek.go.com/Titles?qt=",
                     "&nh="),
      new ParametryWyszukiwania("lycos",
                     "http://lycospro.lycos.com/cgi-bin/" +
                        "pursuit?query=",
                     "&maxhits="),
      new ParametryWyszukiwania("hotbot",
                     "http://www.hotbot.com/?MT=",
                     "&DC=")
    };

  public ParametryWyszukiwania(String nazwa,
                    String adresURL,
                    String znacznikLiczbyWynikow) {
    this.nazwa = nazwa;
    this.adresURL = adresURL;
    this.znacznikLiczbyWynikow = znacznikLiczbyWynikow;
  }

  public String utworzAdresURL(String searchString,
                       String liczbaWynikow) {
    return(adresURL + searchString +
           znacznikLiczbyWynikow + liczbaWynikow);
  }

  public String pobierzNazwe() {
    return(nazwa);
  }

  public static ParametryWyszukiwania[] pobierzGlowneAdresy() {
    return(glowneAdresy);
  }
}
```

Listing 21.3. *RownolegleWyszukiwania.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN">
<HTML>
<HEAD>
  <TITLE>Wyniki równolegle wykonanych zapytań do wielu wyszukiwarek
internetowych</TITLE>
</HEAD>
```

```
<FRAMESET ROWS="120,*">
  <FRAME SRC="RamkaApletu.html" SCROLLING="NO">
  <FRAMESET COLS="*,*,*">
    <FRAME SRC="RamkaGoogle.html" NAME="results0">
    <FRAME SRC="RamkaInfoseek.html" NAME="results1">
    <FRAME SRC="RamkaLycos.html" NAME="results2">
  </FRAMESET>
</FRAMESET>
```

Listing 21.4. *RamkaApletu.html*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Ramka apletu służącego do wyszukiwania informacji</TITLE>
</HEAD>

<BODY BGCOLOR="WHITE">
<CENTER>
<APPLET CODE="ApletWyszukiwania.class" WIDTH=600 HEIGHT=100>
  <B>Ten przykład wymaga użycia przeglądarki obsługującej język Java.</B>
</APPLET>
</CENTER>
</BODY>
</HTML>
```

21.3. Wykorzystywanie metody GET oraz przetwarzanie wyników w sposób bezpośredni (tunelowanie HTTP)

W poprzednim przykładzie aplet informował przeglądarkę o konieczności wyświetlenia w określonej ramce danych wynikowych pochodzących z programu działającego po stronie serwera. Wykorzystywanie przeglądarki do wyświetlania wyników jest całkiem rozsądne podczas korzystania z już działających usług, ponieważ większość programów CGI została skonfigurowana w celu zwracania wyników w postaci dokumentów HTML. Tym niemniej, jeśli tworzy się zarówno aplikację klienta, jak też część obsługującą proces po stronie serwera, marnotrawstwem wydaje się wysyłanie wyników w postaci całego dokumentu HTML. W niektórych wypadkach ogromnym udogodnieniem byłaby możliwość wysyłania danych do już działającego apletu. Aplet mógłby następnie zaprezentować otrzymane dane w formie wykresu lub w innej dowolnej postaci. Takie podejście niekiedy jest nazywane tunelowaniem HTTP, ponieważ własny protokół używany do komunikacji osadzony jest wewnątrz pakietów HTTP stosowanych podczas przekazywania przez serwery pośredniczące szyfrowania, przekierowywania przez serwer, przekazywania przez zapory sieciowe itp.

Istnieją dwie główne odmiany takiej metody. Obie wykorzystują klasę `URLConnection` w celu otwarcia strumienia wejściowego przy użyciu danego adresu URL. Różnica tkwi w rodzaju wykorzystywanego strumienia. Pierwszą możliwością jest użycie strumienia `BufferedInpu-`

tStream lub innego strumienia niskopoziomowego, pozwalającego na odczytanie danych binarnych lub danych w postaci kodu ASCII, pochodzących z dowolnego programu działającego po stronie serwera. Ta metoda zostanie opisana w pierwszym punkcie tego podrozdziału. Druga możliwość polega na wykorzystaniu strumienia ObjectInputStream w celu bezpośredniego odczytania wysokopoziomowych struktur danych. Zostanie ona opisana w drugim punkcie tego podrozdziału. Metoda ta jest możliwa do wykorzystania jedynie wtedy, gdy program działający po stronie serwera jest również napisany przy użyciu języka programowania Java.

Odczytywanie danych binarnych oraz danych tekstowych

Aplet może odczytywać dane przesłane przez serwer. W tym celu musi zostać najpierw utworzony obiekt klasy URLConnection, używającej adresu URL programu działającego po stronie serwera. Następnie musi dołączony zostać do niego strumień klasy BufferedInputStream. W celu zaimplementowania tej metody w aplikacji klienta konieczne jest wykonanie siedmiu głównych czynności, opisanych po kolei poniżej. W przedstawionym opisie pominięty został kod znajdujący się po stronie serwera, ponieważ zdefiniowany kod klienta działa z dowolnym programem pracującym na serwerze lub ze statyczną stroną WWW.

Zwróć uwagę, że wiele operacji na strumieniach generuje wyjątek klasy IOException, dlatego też przedstawione poniżej instrukcje muszą być umieszczone w bloku try … catch.

1. **Utworzenie obiektu URL wskazującego na domowy serwer apletu**
   Do konstruktora URL można przekazać bezwzględny adres URL (na przykład w postaci *http://serwer/sciezka*), jednak, z tego względu, że ograniczenia spowodowane zabezpieczeniami uniemożliwiają połączenia wykonywane z apletów do serwerów innych niż ten, na którym są one umieszczone, bardziej sensowne wydaje się utworzenie adresu URL bazującego na nazwie serwera, z którego został on pobrany.

   ```
   URL biezacaStrona = getCodeBase();
   String protokol = biezacaStrona.getProtocol();
   String serwer = biezacaStrona.getHost();
   int port = biezacaStrona.getPort();
   String przyrostekAdresu = "/servlet/PewienSerwlet";
   URL adresURL = new URL(protokol, serwer, port, przyrostekAdresu);
   ```

2. **Utworzenie obiektu URLConnection** Metoda openConnection, zdefiniowana w klasie URL, powoduje zwrócenie obiektu URLConnection. Obiekt ten będzie następnie wykorzystany w celu pobrania strumieni wykorzystywanych do komunikacji.

   ```
   URLConnection polaczenie = adresURL.openConnection();
   ```

3. **Nakazanie przeglądarce, aby nie umieszczała danych dołączonych do adresu URL w swojej pamięci podręcznej** Pierwszą czynnością konieczną do wykonania, wykorzystując obiekt URLConnection jest wskazanie, aby dane przekazywane za jego pomocą nie były nigdy umieszczane przez przeglądarkę w pamięci podręcznej. Tego rodzaju podejście gwarantuje uzyskanie za każdym razem najświeższych wyników.

   ```
   polaczenie.setUseCaches(false);
   ```

4. **Ustawienie wszystkich pożądanych nagłówków HTTP** Jeśli chce się ustawić nagłówki żądania HTTP (patrz podrozdział 19.7.), można wykorzystać metodę o nazwie `setRequestProperty`.

```
polaczenie.setRequestProperty("naglowek", "wartosc");
```

5. **Utworzenie strumienia wejściowego** Istnieje szereg odpowiednich strumieni, lecz najpowszechniej wykorzystywanym jest strumień `BufferedReader`. Połączenie z serwerem WWW ustanawiane jest w chwili tworzenia strumienia wejściowego.

```
BufferedReader strumienWejsciowy = new BufferedReader(new
InputStreamReader(polaczenie.getInputStream()));
```

6. **Odczytanie wszystkich wierszy dokumentu** Specyfikacja HTTP wymaga zamknięcia przez serwer połączenia po zakończeniu wysyłania dokumentu. W wypadku zamknięcia połączenia, użycie metody o nazwie `readLine` powoduje zwrócenie wartości `null`. Dlatego też należy zwyczajnie odczytywać dane wejściowe, aż do czasu uzyskania wartości `null`.

```
String wiersz;
while ((wiersz = strumienWejsciowy.readLine()) != null) {
  zrobCosPrzyUzyciu(wiersz);
}
```

7. **Zamknięcie strumienia wejściowego**

```
strumienWejsciowy.close();
```

Odczytywanie struktur danych zachowanych przy użyciu serializacji

Przedstawione w poprzednich rozdziałach pomysły prezentowały sposoby wymiany informacji pomiędzy apletem a dowolnym programem działającym po stronie serwera lub też ze statyczną stroną WWW. Niemniej jednak, kiedy aplet komunikuje się z serwletem, istnieje jeszcze lepszy sposób wymiany danych. Zamiast wysyłania ich w postaci binarnej lub tekstowej, serwlet może przesyłać dowolne struktury danych używając w tym celu mechanizmu serializacji, udostępnianego standardowo przez język Java. Aplet następnie może odczytać te dane za pomocą jednego wywołania metody `readObject`. Nie wymagana jest w takim wypadku żmudna analiza danych. Poniżej przedstawione zostały czynności konieczne do implementacji tunelowania HTTP

Aplikacja działająca po stronie klienta

W celu odczytania struktur danych, zachowanych przy użyciu mechanizmu serializacji, przesyłanych przez serwlet, aplet musi wykonać siedem poniżej przedstawionych czynności. Jedynie czynności przedstawione w punktach 5. oraz 6. różnią się od tych, wymaganych podczas odczytywania danych tekstowych opisanych poprzednio. Opis czynności został znacznie uproszczony, dzięki pominięciu bloków `try … catch`.

1. **Utworzenie obiektu URL wskazującego na domowy serwer apletu** Podobnie jak poprzednio, ponieważ adres URL musi wskazywać na serwer, z którego pobrany został aplet, bardziej sensowne jest określenie adresu względnego i pozwolenie na skonstruowanie go w sposób automatyczny.

```
URL biezacaStrona = getCodeBase();
String protokol = biezacaStrona.getProtocol();
String serwer = biezacaStrona.getHost();
int port = biezacaStrona.getPort();
String przyrostekAdresu = "/servlet/PewienSerwlet";
URL adresURL = new URL(protokol, serwer, port, przyrostekAdresu);
```

2. **Utworzenie obiektu URLConnection** Metoda `openConnection`, zdefiniowana w klasie URL, powoduje zwrócenie obiektu `URLConnection`. Obiekt ten będzie następnie wykorzystany w celu pobrania strumieni wykorzystywanych do komunikacji.

```
URLConnection polaczenie = adresURL.openConnection();
```

3. **Nakazanie przeglądarce, aby nie umieszczała danych dołączonych do adresu URL w swojej pamięci podręcznej** Pierwszą czynnością konieczną do wykonania jest wskazanie, dzięki wykorzystaniu obiektu `URLConnection`, aby dane przekazywane za jego pomocą nie były nigdy umieszczane przez przeglądarkę w pamięci podręcznej. Tego rodzaju podejście gwarantuje uzyskanie najświeższych wyników za każdym razem.

```
polaczenie.setUseCaches(false);
```

4. **Ustawienie wszystkich pożądanych nagłówków HTTP** Jeśli chce się ustawić nagłówki żądania HTTP (patrz podrozdział 19.7), można w tym celu użyć metody o nazwie `setRequestProperty`.

```
polaczenie.setRequestProperty("naglowek", "wartosc");
```

5. **Utworzenie strumienia wejściowego klasy ObjectInputStream** Konstruktor tej klasy pobiera strumień danych nieprzetworzonych, udostępniany przez obiekt klasy `URLConnection`. Połączenie z serwerem WWW ustanawiane jest w momencie utworzenia strumienia wejściowego.

```
ObjectInputStream strumienWejsciowy = new
   ObjectInputStream(polaczenie.getInputStream());
```

6. **Odczytanie struktur danych przy użyciu metody readObject** Metoda ta zwraca obiekt klasy `Object`, dlatego też konieczne jest wykonanie rzutowania na bardziej określoną klasę obiektu przesyłanego w rzeczywistości przez serwer.

```
PewnaKlasa wartosc = (PewnaKlasa)strumienWejsciowy.readObject();
  zrobCosPrzyUzyciu(wartosc);
```

7. **Zamknięcie strumienia wejściowego**

```
strumienWejsciowy.close();
```

Aplikacja działająca po stronie serwera

Aby wysłać do apletu struktury danych zachowane przy użyciu serializacji, serwlet musi wykonać cztery poniżej przedstawione czynności. Założono, że metody `doGet` oraz `doPost` dysponują przekazanymi w postaci argumentów obiektami `request` oraz `response` klas `HttpServletRequest` oraz `HttpServletResponse`. Podobnie jak poprzednio, przedstawione czynności zostały uproszczone przez pominięcie wymaganych bloków `try … catch`.

1. **Zasygnalizowanie przesyłania danych binarnych** Jako typ MIME przesyłanej odpowiedzi konieczne jest określenie `application/x-java-serialized-object`. Jest to standardowy typ MIME wskazujący obiekty zakodowane za pomocą strumienia `ObjectOutputStream`. Jednak w praktyce, z tego względu, że to aplet (a nie przeglądarka) odczytuje dane, określenie tego typu nie jest tak bardzo ważne. Więcej informacji na temat typów MIME zostało przedstawionych w podrozdziale 19.10, „Odpowiedź serwera: Nagłówki odpowiedzi HTTP".

```
String rodzajZawartosci = "application/x-java-serialized-object";
response.setContentType(rodzajZawartosci);
```

2. **Utworzenie obiektu klasy ObjectOutputStream**

```
ObjectOutputStream daneWyjsciowe = new
ObjectOutputStream(response.getOutputStream());
```

3. **Zapisanie struktury danych wykorzystując metodę writeObject** Większość, obsługiwanych wewnętrznie przez język Java, struktur danych może być wysłanych przy użyciu metody `writeObject`. Jednakże klasy utworzone samodzielnie muszą implementować interfejs `Serializable`. To dość proste do spełnienia wymaganie, ponieważ interfejs ten nie definiuje żądanych metod. Zwyczajnie należy zadeklarować klasę implementującą wspomniany interfejs.

```
PewnaKlasa wartosc = new PewnaKlasa(...);
daneWyjsciowe.writeObject(wartosc);
```

4. **Opróżnienie strumienia w celu zagwarantowania wysłania całej zawartości do aplikacji klienta**

```
daneWyjsciowe.flush();
```

W kolejnym podrozdziale przedstawimy przykład zastosowania tunelowania HTTP.

21.4. Przeglądarka żądań wysyłanych do serwera, używająca serializacji obiektów oraz tunelowania HTTP

Wielu ludzi interesuje się rodzajem zapytań wysyłanych do najpopularniejszych wyszukiwarek internetowych. Wynika to po części z ciekawości („Czy naprawdę aż 64 procent zapytań zadanych w wyszukiwarce AltaVista pochodzi od pracodawców poszukujących programistów znających technologię Java?"), a po części z dążeń programistów do dostosowania zawartości tworzonych przez nich witryn do najczęściej zadawanych zapytań, co powoduje zazwyczaj zwiększenie ich pozycji w rankingu wyszukiwarek.

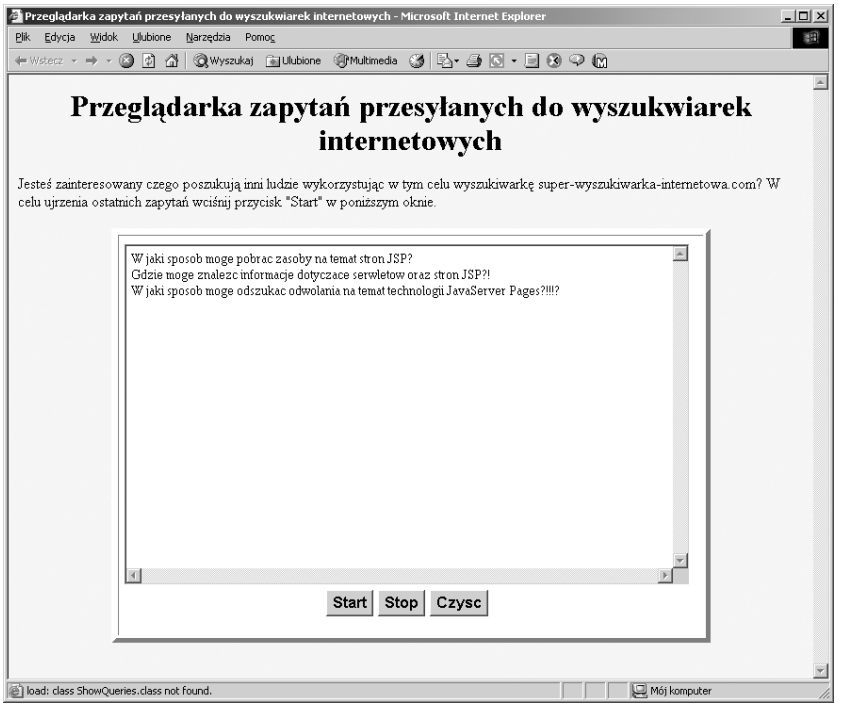
W tym podrozdziale przedstawiono wykorzystanie połączenia aplikacji apletu oraz serwletu, służącego do zaprezentowania fikcyjnej wyszukiwarki o nazwie *super-wyszukiwarka-internetowa.com*, która na bieżąco wyświetla przykładowe zapytania wysyłane przez użytkowników. Na listingu 21.5 pokazany został główny aplet wykorzystujący zewnętrzną klasę

(przedstawioną na listingu 21.6) do pobierania zapytań w wątku działającym w tle. Po zainicjowaniu procesu przez użytkownika, co pół sekundy aplet umieszcza w przewijanym obszarze tekstowym przykładowe zapytanie. Zostało to przedstawione na rysunku 21.3. Na koniec listing 21.7. prezentuje kod aplikacji serwletu tworzącego zapytania po stronie serwera. Powoduje on wygenerowanie losowego przykładu rzeczywistych, zadawanych przez użytkownika zapytań i podczas każdego żądania wysyła do klienta 50 z nich. Niektóre szczegóły dotyczące zastosowanych metod zostały omówione w rozdziale 19.

Osoby, które będą chciały pobrać kod źródłowy apletu oraz serwletu z archiwum umieszczonego pod adresem: *ftp://ftp.helion.pl/przyklady/serinp.zip* i będą próbowały uruchomić aplikację samodzielnie, muszą wiedzieć, że będzie ona działać jedynie wtedy, gdy wyświetlona zostanie główna strona HTML przy wykorzystaniu protokołu HTTP (na przykład używając w tym celu adresu URL postaci *http://...*). Pobranie strony bezpośrednio z dysku zakończy się niepowodzeniem, ponieważ aplet w celu komunikowania się z serwletem łączy się ze swoim katalogiem domowym. Poza tym, mówiąc ogólnie, próba nawiązania połączenia przy użyciu obiektu `URLConnection` w wypadku apletów, które nie korzystają z połączenia HTTP, nie powiedzie się.

Listing 21.5. *PokazZapytania.java*

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

/** Aplet odczytujący tablicę łańcuchów znakowych
 *  utworzoną przez obiekt klasy QueryCollection i umieszczający ją w obszarze
 *  TextArea. Klasa QueryCollection przesyła dane przy użyciu serializacji.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class PokazZapytania extends Applet
                        implements ActionListener, Runnable {
  private TextArea obszarZapytan;
  private Button przyciskStart, przyciskStop, przyciskCzysc;
  private KolekcjaZapytan biezaceZapytania;
  private KolekcjaZapytan kolejneZapytania;
  private boolean isUruchomiony = false;
  private String adres =
    "/servlet/cwp.GeneratorZapytan";
  private URL biezacaStrona;

  public void init() {
    setBackground(Color.white);
    setLayout(new BorderLayout());
    obszarZapytan = new TextArea();
    obszarZapytan.setFont(new Font("Serif", Font.PLAIN, 14));
    add(obszarZapytan, BorderLayout.CENTER);
    Panel panelPrzyciskow = new Panel();
```



Rysunek 21.3. *Wynik działania apletu PokazZapytania*

```
  Font czcionkaPrzycisku = new Font("SansSerif", Font.BOLD, 16);
  przyciskStart = new Button("Start");
  przyciskStart.setFont(czcionkaPrzycisku);
  przyciskStart.addActionListener(this);
  panelPrzyciskow.add(przyciskStart);
  przyciskStop = new Button("Stop");
  przyciskStop.setFont(czcionkaPrzycisku);
  przyciskStop.addActionListener(this);
  panelPrzyciskow.add(przyciskStop);
  przyciskCzysc = new Button("Czysc");
  przyciskCzysc.setFont(czcionkaPrzycisku);
  przyciskCzysc.addActionListener(this);
  panelPrzyciskow.add(przyciskCzysc);
  add(panelPrzyciskow, BorderLayout.SOUTH);
  biezacaStrona = getCodeBase();
  // Żąda zestawu przykładowych zapytań. Są one pobierane
  // przy użyciu wątku działającego w tle, zaś aplet sprawdza, czy jego działanie
  // zostało zakończone
  // przed podjęciem próby pobrania łańcuchów znakowych.
  biezaceZapytania = new KolekcjaZapytan(adres, biezacaStrona);
  kolejneZapytania = new KolekcjaZapytan(adres, biezacaStrona);
}

/** Jeśli zostanie wciśnięty przycisk "Start" system
 *  uruchomi wątek działający w tle wyświetlający zapytania w obszarze
 *  tekstowym. Wciśnięcie "Stop"
 *  zatrzymuje działanie procesu, zaś "Czyść" opróżnia
 *  obszar tekstowy.
 */
```

```
public void actionPerformed(ActionEvent event) {
  if (event.getSource() == przyciskStart) {
    if (!isUruchomiony) {
      Thread procesWyswietlajacy = new Thread(this);
      isUruchomiony = true;
      obszarZapytan.setText("");
      procesWyswietlajacy.start();
      showStatus("Proces wyświetlający zapytania został uruchomiony...");
    } else {
      showStatus("Proces wyświetlający zapytania już działa...");
    }
  } else if (event.getSource() == przyciskStop) {
    isUruchomiony = false;
    showStatus("Proces wyświetlający zapytanai został zatrzymany...");
  } else if (event.getSource() == przyciskCzysc) {
    obszarZapytan.setText("");
  }
}

/** Proces działający w tle pobiera obiekt biezaceZapytania
 *  i co pół sekundy umieszcza jedno z przechowywanych w nim zapytań
 *  na dole obszaru tekstowego. Po wyświetleniu wszystkich zapytań,
 *  wątek kopiuje zawartość obiektu
 *  kolejneZapytania  do obiektu
 *  biezaceZapytania, wysyła nowe żądanie do serwera
 *  w celu ponownego zapełnienia obiektu kolejneZapytania i powtarza cały proces
 */

public void run() {
  while(isUruchomiony) {
    pokazZapytania(biezaceZapytania);
    biezaceZapytania = kolejneZapytania;
    kolejneZapytania = new KolekcjaZapytan(adres, biezacaStrona);
  }
}

private void pokazZapytania(KolekcjaZapytan zbiorZapytan) {
  // Jeśli żądanie zostało wysłane do serwera, lecz nie zwrócił on jeszcze wyników,
  // wtedy należy sprawdzać co sekundę, czy zadanie zostało ukończone.
  // Powinno zdarzać się dość rzadko, lecz może występować w przypadku bardzo wolnych
  // połączeń sieciowych lub przeciążonych serwerów.
  while(!zbiorZapytan.isKoniec()) {
    showStatus("Oczekiwanie na dane z serwera...");
    pauza(1);
  }
  showStatus("Otrzymano dane z serwera...");
  String[] zapytania = null;
  try {
     zapytania = zbiorZapytan.pobierzZapytania();
  } catch (IOException e) {}
  String znakkoncawiersza = "\n";
  // umieść nowy ciąg zapytania w obszarze tekstowym co pół sekundy.
  for(int i=0; i<zapytania.length; i++) {
    if (!isUruchomiony) {
      return;
    }
    obszarZapytan.append(zapytania[i]);
    obszarZapytan.append(znakkoncawiersza);
```

```
      pauza(0.5);
    }
  }

  public void pauza(double seconds) {
    try {
      Thread.sleep((long)(seconds*1000));
    } catch(InterruptedException ie) {}
  }
}
```

Listing 21.6. *KolekcjaZapytan.java*

```
import java.net.*;
import java.io.*;

/** Po utworzeniu tej klasy, zwraca ona natychmiast wartość,
 *  w przypadku metody getZapytania jest to wartość null, za to w przypadku
isZakonczone wartość false
 *  W międzyczasie zostaje utworzony wątek służący do pobrania ciągów zapytań z serwera
 *  odczytywanych za jednym zamachem przy użyciu obiektu klasy
 *  ObjectInputStream. Po pobraniu ciągów zapytań są one umieszczane w zmiennej
 *  odczytywanej przez metodę getZapytania,
 *  zaś wartość znacznika isZakonczone zostaje zmieniona na true.
 *  Klasa wykorzystywana przez aplet ShowQueries.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class KolekcjaZapytan implements Runnable {
  private String[] zapytania;
  private String[] tymczasoweZapytania;
  private boolean isKoniec = false;
  private URL daneAdresuURL;

  public KolekcjaZapytan(String przyrostekURL, URL biezacaStrona) {
    try {
      // Konieczne jest określenie jedynie przyrostka adresu URL, jako że
      // dalsza część adresu określana jest na podstawie bieżącej strony.
      String protokol = biezacaStrona.getProtocol();
      String host = biezacaStrona.getHost();
      int port = biezacaStrona.getPort();
      daneAdresuURL = new URL(protokol, host, port, przyrostekURL);
      Thread watekPobierajacyZapytania = new Thread(this);
      watekPobierajacyZapytania.start();
    } catch(MalformedURLException mfe) {
      isKoniec = true;
    }
  }

  public void run() {
    try {
      tymczasoweZapytania = pobierzZapytania();
```

```
      zapytania = tymczasoweZapytania;
    } catch(IOException ioe) {
      tymczasoweZapytania = null;
      zapytania = null;
    }
    isKoniec = true;
  }

  public String[] getZapytania() {
    return(zapytania);
  }

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

  public String[] pobierzZapytania() throws IOException {
    URLConnection polaczenie = daneAdresuURL.openConnection();
    // Należy się upewnić, że przeglądarka nie umieszcza tego adresu URL w pamięci podręcznej
    // Pragniemy przecież uzyskać różne zapytania dla każdego żądania.
    polaczenie.setUseCaches(false);
    // Należy użyć ObjectInputStream tak, aby możliwe było jednorazowe odczytanie String[]
    ObjectInputStream in =
      new ObjectInputStream(polaczenie.getInputStream());
    try {
      // metoda readObject zwraca obiekt,
      // dlatego konieczne jest zrzutowanie na rzeczywisty obiekt
      String[] ciagiZapytan = (String[])in.readObject();
      return(ciagiZapytan);
    } catch(ClassNotFoundException cnfe) {
      return(null);
    }
  }
}
```

Listing 21.7. *GeneratorZapytan.java*

```
package cwp;

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

/** Serwlet tworzący tablice łańcuchów znakowych z zapytaniami
 *  i wysyłający je przy użyciu obiektu klasy ObjectOutputStream do apletu
 *  lub innego klienta napisanego w jezyku Java.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class GeneratorZapytan extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
```

```
    throws ServletException, IOException {
  boolean uzyjNumerowania = true;
  String znacznikNumerowania =
    request.getParameter("useNumbering");
  if ((znacznikNumerowania == null) ||
      znacznikNumerowania.equals("false")) {
    uzyjNumerowania = false;
  }
  String typZawartosci =
    "application/x-java-serialized-object";
  response.setContentType(typZawartosci);
  ObjectOutputStream out =
    new ObjectOutputStream(response.getOutputStream());
  String[] zapytania = getZapytania(uzyjNumerowania);
  // Jeśli wysyła się niestandardowe struktury danych, należy upewnić się, że
  // określona została implementacja interfejsu serializacji "implements Serializable".
  out.writeObject(zapytania);
  out.flush();
}

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

private String[] getZapytania(boolean uzyjNumerowania) {
  String[] zapytania = new String[50];
  for(int i=0; i<zapytania.length; i++) {
    zapytania[i] = losoweZapytanie();
    if (uzyjNumerowania) {
      zapytania[i] = "" + (i+1) + ": " + zapytania[i];
    }
  }
  return(zapytania);
}

// Rzeczywiste zapytania wysyłane przez ludzi :-)

private String losoweZapytanie() {
  String[] polozenie = { "Gdzie  ", "W jaki sposob " };
  String[] czynnosc =
    { "moge odszukac ", "moge znalezc ", "moge pobrac " };
  String[] zrodlo =
    { "informacje ", "zasoby ", "dane ", "odwolania " };
  String[] przyimek = { "na temat ", "odnoszace sie do ", "dotyczace " };
  String[] temat =
    { "ksiazki Serwisy internetowe.Programowanie",
      "tekstu opisujacego serwlety oraz strony JSP",
      "serwletow oraz stron JavaServer Pages",
      "serwletow oraz stron JSP",
      "ksiazki Serwisy internetowe.Programowanie (Wydanie 2)",
      " Serwisy internetowe.Programowanie (Wydanie 2)",
      "programowania serwletow", "technologii JavaServer Pages", "stron JSP",
      "alternatywny metod programowania CGI przy uzyciu jezyka Java", "programow
      w jezyku Java dzialajacych po stronie serwera" };
  String[] zakonczenie = { "?", "?", "?", "?!", "?!!!?" };
  String[][] szablonyZdan =
```

```
        { polozenie, czynnosc, zrodlo,
          przyimek, temat, zakonczenie };
      String zapytanie = "";
      for(int i=0; i<szablonyZdan.length; i++) {
        zapytanie = zapytanie + losowyElement(szablonyZdan[i]);
      }
      return(zapytanie);
    }

    private String losowyElement(String[] lancuchy) {
      int indeks = (int)(Math.random()*lancuchy.length);
      return(lancuchy[indeks]);
    }
  }
```

21.5. Wysyłanie danych przy użyciu metody POST i przetwarzanie wyników wewnątrz apletu (tunelowanie HTTP)

Podczas przesyłania danych przy użyciu metody GET, aplet ma dwie możliwości wyświetlenia wyników. Może on nakazać wykonanie tej czynność przeglądarce (utworzyć odpowiedni obiekt adresu URL, a następnie wywołać metodę AppletContext().showDocument) lub też przetworzyć dane samodzielnie (utworzyć obiekt adresu URL, pobrać obiekt klasy URLConnection, otworzyć strumień wejściowy i odczytać dane). Metody te zostały omówione, odpowiednio, w podrozdziale 21.1. oraz 21.3. W przypadku wykorzystywania danych POST możliwe jest wykorzystanie tylko drugiej metody, ponieważ konstruktor obiektu URL nie zawiera metody pozwalającej na skojarzenie z nim danych przesyłanych przy użyciu metody POST. Przekazywanie danych w ten sposób ma kilka zalet, ale też niedogodności w porównaniu z używaniem metody GET. Do dwóch najważniejszych niedogodności należą: konieczność umieszczenia programu działającego po stronie serwera na tym samym komputerze co aplet oraz konieczność wyświetlania danych przy użyciu apletu. Nie istnieje bowiem możliwość przekazania kodu HTML do przeglądarki. Do zalet natomiast należy to, że program działający po stronie serwera może być prostszy (nie trzeba umieszczać wyników w kodzie HTML), zaś aplet może odświeżać wyświetlane dane bez potrzeby ponownego odczytywania strony. Co więcej, aplety wykorzystujące do komunikacji metodę POST, mogą stosować w celu wysyłania danych do serwletu strumienie przekazujące dane zapisane za pomocą serializacji, oprócz już opisanej wcześniej, analogicznej metody podczas pobierania danych z serwletu. To poważny plus, ponieważ wykorzystywanie danych serializowanych oraz tunelowania HTTP pozwala na ominięcie zabezpieczeń stosowanych na zaporach sieciowych, aby uniemożliwić bezpośrednie połączenia pomiędzy gniazdami. Aplety wykorzystujące metodę GET mogą odczytywać dane zapisywane w ten sposób (patrz podrozdział 21.4), lecz nie są w stanie ich wysyłać, gdyż niemożliwe jest dołączenie do adresów URL dowolnych danych binarnych.

Aby umożliwić wysyłanie z apletów do serwera danych przy użyciu metody `POST` oraz odczytywanie wyników, wymagane jest wykonanie trzynastu czynności przedstawionych poniżej. Chociaż ich liczba jest dość duża, to jednak każda z czynności jest względnie prosta. Kod został znacznie uproszczony dzięki pominięciu bloków postaci `try … catch`, w których powinny zostać umieszczone wszystkie instrukcje.

1. **Utworzenie obiektu URL wskazującego na domowy serwer apletu** Podobnie jak poprzednio, ze względu na to, że adres URL musi wskazywać na serwer, z którego pobrany został aplet, bardziej sensowne jest określenie adresu względnego i pozwolenie na skonstruowanie go w sposób automatyczny.

```
URL biezacaStrona = getCodeBase();
String protokol = biezacaStrona.getProtocol();
String serwer = biezacaStrona.getHost();
int port = biezacaStrona.getPort();
String przyrostekAdresu = "/servlet/PewienSerwlet";
URL adresURL = new URL(protokol, serwer, port, przyrostekAdresu);
```

2. **Utworzenie obiektu URLConnection** Obiekt ten będzie następnie wykorzystany w celu pobrania strumieni wykorzystywanych do komunikacji.

```
URLConnection polaczenie = adresURL.openConnection();
```

3. **Nakazanie przeglądarce, aby nie umieszczała danych dołączonych do adresu URL w swojej pamięci podręcznej**

```
polaczenie.setUseCaches(false);
```

4. **Uzyskanie od serwera pozwolenia na wysyłanie danych, a nie jedynie odbieranie wyników**

```
polaczenie.setDoOutput(true);
```

5. **Utworzenie strumienia ByteArrayOutputStream, wykorzystywanego do buforowania danych, które będą wysyłane do serwera**

   Strumień ByteArrayOutputStream wykorzystywany jest w celu określenia rozmiaru danych wyjściowych, tak aby aplet był w stanie wysłać nagłówek `Content-Length`, wymagany podczas żądań typu `POST`. Konstruktor klasy `ByteArrayOutputStream` wskazuje początkowy rozmiar bufora danych, lecz jego podanie nie jest wymagane, gdyż rozmiar bufora będzie zwiększany w razie potrzeby w sposób automatyczny.

```
ByteArrayOutputStream stumienBinarny = new ByteArrayOutputStream(512);
```

6. **Skojarzenie strumienia wyjściowego z obiektem klasy ByteArrayOutputStream**

   Wysłając normalne dane formularza, należy wykorzystać obiekt klasy `PrintWriter`. W celu wysłania struktur danych zachowanych przy użyciu serializacji, trzeba wykorzystać w zamian obiekt klasy `ObjectOutputStream`.

```
PrintWriter daneWyjsciowe = new PrintWriter(strumienBinarny, true);
```

7. **Umieszczenie danych w buforze** W przypadku normalnych danych formularza należy wykorzystać metodę `print`. Natomiast w celu wysłania obiektów wysokiego poziomu, zachowanych przy użyciu serializacji, metodę `writeObject`.

```
String war1 = URLEncoder.encode(PewnaWartosc1);
String war2 = URLEncoder.encode(PewnaWartosc2);
String dane = "param1=" + war1 + "&param2=" + war2;  //Zwroc uwage na znak &
daneWyjsciowe.print(dane); //zwróć uwagę na użycie metody print, a nie println
daneWyjsciowe.flush(); //użycie tej metody jest konieczne ponieważ nie została
wykorzystana metoda println
```

8. **Ustawienie nagłówka o nazwie Content-Length** Nagłówek ten jest wymagany w przypadku danych przesyłanych przy użyciu metody `POST`, nawet jeśli nie jest on używany w przypadku metody `GET`.

```
polaczenie.setRequestProperty("Content-Length",
   String.valueOf(strumienBinarny.size()));
```

9. **Ustawienie nagłówka o nazwie Content-Type** Przeglądarka Netscape używa domyślnie typu `multipart.form-data`, jednak przesyłanie regularnych danych formularza wymaga ustawienia nagłówka o wartości `application/x-www-form-urlencoded`, który z kolei jest domyślny w przypadku przeglądarki Internet Explorer. Dlatego też, w celu umożliwienia przenoszenia programów, w przypadku wysyłania zwykłych danych formularza, wartość ta powinna zostać ustawiona w sposób jawny. W przypadku wysyłania danych zapisanych przy użyciu serializacji wartość ta nie jest istotna.

```
polaczenie.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
```

10. **Wysłanie rzeczywistych danych**

```
strumienBinarny.writeTo(polaczenie.getOutputStream());
```

11. **Utworzenie strumienia wejściowego** W przypadku danych tekstowych (ASCII) oraz binarnych zazwyczaj wykorzystywany jest obiekt klasy `BufferedReader`, natomiast w przypadku danych serializowanych obiekt klasy `ObjectInputStream`.

```
BufferedReader daneWejsciowe = new BufferedReader(new
    InputStreamReader(polaczenie.getInputStream()));
```

12. **Odczytanie wyniku**

    Szczegółowy opis wymaganych czynności zależy od rodzaju danych wysyłanych przez serwer. Poniżej przedstawiony został przykład wykonujący pewne nieokreślone bliżej czynności z każdym wierszem danych przesyłanych przez serwer:

```
String wiersz;
while((wiersz = daneWejsciowe.readLine()) != null) {
  zrobCosPrzyUzyciu(wiersz);
}
```

13. **Pogratulowanie sobie** Procedura obsługi danych przesyłanych przy użyciu metody `POST` jest długa i zawiła. Na szczęście, jest to dość standardowy proces. Oprócz tego można zawsze pobrać kod źródłowy umieszczony pod adresem *ftp://ftp.helion.pl/przyklady/serinp.zip* i wykorzystać go w charakterze szablonu, od którego można rozpocząć tworzenie własnych programów.

W kolejnym podrozdziale przedstawiony zostanie przykład apletu, wykonującego wszystkie opisane czynności.

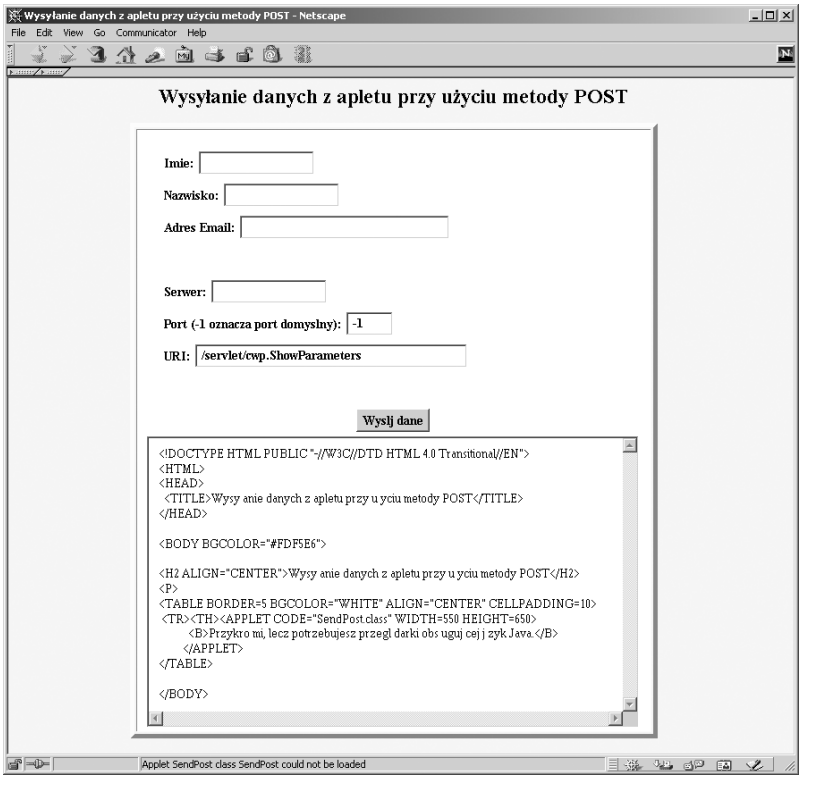
21.6. Aplet wysyłający dane przy użyciu metody POST

Na listingu 21.8. przedstawiony został aplet wykorzystujący metodę opisaną w poprzednim podrozdziale. Aplet ten w celu wysłania danych przy użyciu metody POST pod wskazany adres URL wykorzystuje obiekt klasy `URLConnection` oraz `ByteArrayOutputStream`. Używa on również klasy o nazwie `EtykietowanePoleTekstowe`, która umieszczona została w archiwum znajdującym się pod adresem: *ftp://ftp.helion.pl/przyklady/serinp.zip*.

Na rysunku 21.4. przedstawiony został wynik przesłania danych do serwletu o nazwie `PokazNaglowki`, który jest niezwykle prosty i tworzy stronę WWW, zawierającą wszystkie parametry przesłanego żądania. Szczegółowe informacje na ich temat umieszczone zostały w podrozdziale 19.6, „Żądanie klienta: Dostęp do danych formularza".

Listing 21.8. *WyslijPost.java*

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

/** Aplet odczytujący parametry o nazwach firstName, lastName oraz
 *  emailAddress, a następnie wysyłający je do programu umieszczonego
 *  na wskazanym serwerze, porcie oraz adresie URI przy użyciu metody POST.
 *  <P>
 * Kody zaczerpnięte z polskiej edycji
 * książki Serwisy internetowe. Programowanie
 * Wydawnictwo Helion, Gliwice,
 * ftp://ftp.helion.pl/przyklady/serinp.zip
 * 2001 Marty Hall and Larry Brown;
 */

public class WyslijPost extends Applet
                      implements ActionListener {
  private EtykietowanePoleTekstowe poleImienia, poleNazwiska,
                           poleAdresuEmail, poleNazwySerwera,
                           poleNumeruPortu, poleAdresuURI;
  private Button przyciskWyslij;
  private TextArea obszarWyniku;
  URL biezacaStrona;

  public void init() {
    setBackground(Color.white);
    setLayout(new BorderLayout());
    Panel panelWejsciowy = new Panel();
    panelWejsciowy.setLayout(new GridLayout(9, 1));
    panelWejsciowy.setFont(new Font("Serif", Font.BOLD, 14));
    poleImienia =
      new EtykietowanePoleTekstowe("Imie:", 15);
    panelWejsciowy.add(poleImienia);
    poleNazwiska =
```



Rysunek 21.4. *Wynik działania apletu o nazwie WyslijPost, wysyłającego przy użyciu metody POST dane do serwletu PokazNaglowki*

```
  new EtykietowanePoleTekstowe("Nazwisko:", 15);
panelWejsciowy.add(poleNazwiska);
poleAdresuEmail =
  new EtykietowanePoleTekstowe("Adres Email:", 30);
panelWejsciowy.add(poleAdresuEmail);
Canvas separator1 = new Canvas();
panelWejsciowy.add(separator1);
poleNazwySerwera =
  new EtykietowanePoleTekstowe("Serwer:", 15);

// Aplety wczytywane poprzez sieć mogą komunikować się jedynie z serwerem,
// z którego zostały pobrane.
poleNazwySerwera.podajPoleTekstowe().setEditable(false);

biezacaStrona = getCodeBase();
// metoda getHost zwraca pusty łańcuch znakowy w przypadku apletów wczytanych
z dysku lokalnego.
String serwer = biezacaStrona.getHost();
String komunikatWynikowy = "W tym miejscu pokazany zostanie komunikat...";
if (serwer.length() == 0) {
  komunikatWynikowy = "Bład: Ten aplet musi zostac wczytany \n" +
                  "z rzeczywistego serwera WWW przy użyciu protokołu HTTP, \n" +
                  "a nie z lokalnego dysku za pomocą określenia adresu URL\n" +
                 "postaci 'file:'. Możliwe jest jednak uruchomienie serwera WWW\n" +
                  "na lokalnym serwerze.";
  setEnabled(false);
}
```

```
    poleNazwySerwera.podajPoleTekstowe().setText(serwer);
    panelWejsciowy.add(poleNazwySerwera);
    poleNumeruPortu =
      new EtykietowanePoleTekstowe("Port (-1 oznacza port domyslny):", 4);
    String portString = String.valueOf(biezacaStrona.getPort());
    poleNumeruPortu.podajPoleTekstowe().setText(portString);
    panelWejsciowy.add(poleNumeruPortu);
    poleAdresuURI =
      new EtykietowanePoleTekstowe("URI:", 40);
    String domyslnyAdresURI = "/servlet/cwp.ShowParameters";
    poleAdresuURI.podajPoleTekstowe().setText(domyslnyAdresURI);
    panelWejsciowy.add(poleAdresuURI);
    Canvas separator2 = new Canvas();
    panelWejsciowy.add(separator2);
    przyciskWyslij = new Button("Wyslj dane");
    przyciskWyslij.addActionListener(this);
    Panel buttonPanel = new Panel();
    buttonPanel.add(przyciskWyslij);
    panelWejsciowy.add(buttonPanel);
    add(panelWejsciowy, BorderLayout.NORTH);
    obszarWyniku = new TextArea();
    obszarWyniku.setFont(new Font("Monospaced", Font.PLAIN, 14));
    obszarWyniku.setText(komunikatWynikowy);
    add(obszarWyniku, BorderLayout.CENTER);
  }

  public void actionPerformed(ActionEvent event) {
    try {
      String protokol = biezacaStrona.getProtocol();
      String serwer = poleNazwySerwera.podajPoleTekstowe().getText();
      String ciagPortu = poleNumeruPortu.podajPoleTekstowe().getText();
      int port;
      try {
        port = Integer.parseInt(ciagPortu);
      } catch(NumberFormatException nfe) {
        port = -1; // domyślny port o numerze 80
      }
      String uri = poleAdresuURI.podajPoleTekstowe().getText();
      URL daneURL = new URL(protokol, serwer, port, uri);
      URLConnection polaczenie = daneURL.openConnection();

      // Należy się upewnić, że przeglądarka nie umieszcza adresu URL w buforze
      pamięci podręcznej.
      polaczenie.setUseCaches(false);

      // Należy nakazać przeglądarce umożliwienie wysyłania danych do serwera.
      polaczenie.setDoOutput(true);

      ByteArrayOutputStream strumienBinarny =
        new ByteArrayOutputStream(512); // Rozszerz bufor w razie potrzeby
      // Strumień zapisujący do bufora
      PrintWriter out = new PrintWriter(strumienBinarny, true);
      String danePost =
        "firstName=" + wartoscZakodowana(poleImienia) +
        "&lastName=" + wartoscZakodowana(poleNazwiska) +
        "&emailAddress=" + wartoscZakodowana(poleAdresuEmail);
```

```
        // Zapisz dane przesyłane za pomocą metody POST do lokalnego bufora danych
        out.print(danePost);
        out.flush(); // Oproznij bufor jawnie poniewaz wykorzystana zostala metoda
        print, a nie println

        // Żądania POST wymagają określenia nagłówka Content-Length
        String ciagDlugosci =
          String.valueOf(strumienBinarny.size());
        polaczenie.setRequestProperty
          ("Content-Length", ciagDlugosci);

        // Przeglądarka Netscape ustawia automatycznie nagłówek Content-Type i nadaje
        mu wartość multipart/form-data
        // Jeśli więc przesyłane są zwykłe dane formularza, musisz zmienić jego wartość
        // na application/x-www-form-urlencoded, co z kolei
        // stanowi wartość domyślną ustawianą przez przeglądarkę Internet Explorer.
        Jeśli przesyła się
        // dane POST zapisane przy użyciu strumienia ObjectOutputStream,
        // wartość nagłówka Content-Type nie jest brana pod uwagę, dlatego też
        // można go pominąć.
        polaczenie.setRequestProperty
          ("Content-Type", "application/x-www-form-urlencoded");

        // Należy zapisać dane POST do rzeczywistego strumienia wyjściowego
        strumienBinarny.writeTo(polaczenie.getOutputStream());

        BufferedReader in =
          new BufferedReader(new InputStreamReader
                             (polaczenie.getInputStream()));
        String wiersz;
        String znakKoncaWiersza = "\n";
        obszarWyniku.setText("");
        while((wiersz = in.readLine()) != null) {
          obszarWyniku.append(wiersz);
          obszarWyniku.append(znakKoncaWiersza);
        }
      } catch(IOException ioe) {
        // Należy zapisać informacje testowe na konsoli Javy
        System.out.println("IOException: " + ioe);
      }
    }

    // Obiekt klasy LabeledTextField jest w rzeczywistosci obiektem klasy Panel
    // z umieszczonym nań obiektami klasy Label oraz
    // TextField. Poniższa metoda pobiera wartość umieszczoną w polu tekstowym,
    // następnie koduje ją przy użyciu metody URL-encoded i zwraca uzyskany wynik.

    private String wartoscZakodowana(EtykietowanePoleTekstowe pole) {
      String wartoscOryginalna = pole.podajPoleTekstowe().getText();
      return(URLEncoder.encode(wartoscOryginalna));
    }

  }
```

21.7. Pomijanie serwera HTTP podczas komunikacji z programem po stronie serwera

Chociaż aplety mogą utworzyć połączenia sieciowe jedynie z tym samym komputerem, z którego zostały wczytane, nie muszą używać w tym celu koniecznie tego samego portu (na przykład portu 80 w przypadku połączeń HTTP). W celu komunikacji z różnymi klientami działającymi na tym samym serwerze mogą one używać zwykłych gniazd, JDBC oraz RMI.

Wykonanie tych czynności w apletach przebiega dokładnie tak samo, jak w przypadku zwykłych programów utworzonych w języku Java, dlatego też można użyć metod wykorzystania gniazd, JDBC oraz RMI, które są już znane, pod warunkiem, że używa się tego samego serwera sieciowego, z którego pobrany został aplet.

21.8. Podsumowanie

Formularze HTML stanowią najprostszy oraz najpowszechniej stosowany interfejs użytkownika programów działających po stronie serwera. Tym niemniej, aplety dostarczają bogatszego zestawu elementów interfejsu użytkownika, umożliwiają utworzenie ekranów z ciągle odświeżaną zawartością oraz upraszczają zdecydowanie przesyłanie dużych złożonych struktur danych. Generalną zasadą jest używanie formularzy wszędzie tam, gdzie jest to tylko możliwe. Jednak wygodnie jest dysponować możliwością użycia apletów w przypadku, gdy formularze HTML stają się zbyt ograniczone.