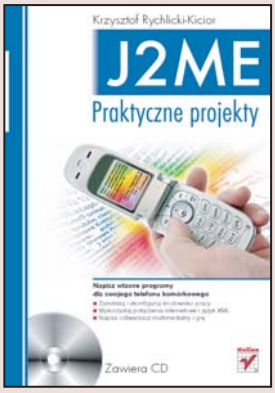
J2ME. Praktyczne projekty

Autor: Krzysztof Rychlicki-Kicior
ISBN: 83-246-0597-5
Format: około 216, stron: B5



Napisz własne programy dla swojego telefonu komórkowego

- Zainstaluj i skonfiguruj środowisko pracy
- Wykorzystaj połączenia internetowe i język XML
- Napisz odtwarzacz multimedialny i grę

Ogromna popularność języka Java wynika między innymi z tego, że napisane w nim programy można uruchomić praktycznie na dowolnym komputerze. Wśród urządzeń „rozumiejących" Javę coraz częściej można znaleźć telefony komórkowe. Java przeznaczona dla urządzeń mobilnych, oznaczona symbolem J2ME, różni się od Javy dla „prawdziwych" komputerów. Tworząc aplikację dla telefonu, należy uwzględnić ograniczony rozmiar pamięci, mniejszą ilość miejsca na wyświetlaczu i inne czynniki. Jednak pomimo to J2ME umożliwia pisanie w pełni funkcjonalnych aplikacji wykorzystujących bazy danych, połączenia sieciowe, technologię XML i usługi sieciowe.

„J2ME. Praktyczne projekty" to podręcznik tworzenia aplikacji dla telefonów komórkowych z wykorzystaniem języka Java. Czytając go, dowiesz się, jakie środowisko programistyczne wybrać i jak dostosować je do wymagań projektów dla urządzeń mobilnych. Nauczysz się tworzyć aplikacje operujące na danych, łączące się z internetem i przetwarzające pliki XML. Napiszesz własny czytnik kanałów RSS, odtwarzacz multimedialny i grę. Poznasz także zasady korzystania z komunikacji Bluetooth w aplikacjach J2ME.

- Konfiguracja środowiska roboczego
- Podstawowe komponenty graficzne w J2ME
- Obsługa wyświetlacza i klawiatury
- Połączenie aplikacji z internetem
- Przetwarzanie plików XML
- Odtwarzanie plików multimedialnych
- Nagrywanie dźwięku
- Tworzenie gier w J2ME
- Obsługa połączeń Bluetooth z poziomu J2ME

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 5.

Obsługa XML w J2ME

W poprzednim rozdziale omówiona została obsługa internetu w J2ME na przykładzie usługi czata. Jedną z najczęściej wykorzystywanych obecnie w internecie technologii jest język XML.

O XML-u pisano już wiele. Jego znajomość jest bardzo ważna przy tworzeniu różnego rodzaju aplikacji; możliwe, że Czytelnik spotkał się już z tym językiem w czasie swojej programistycznej przygody. Omówię go zatem pokrótce, głównie pod kątem projektu tworzonego w tym rozdziale.

XML (ang. *eXtensible Markup Language* — rozszerzalny język znaczników) jest uniwersalnym językiem formalnym służącym do reprezentowania dowolnych danych. Głównymi elementami dokumentu XML są znaczniki (tagi) oraz atrybuty. XML wywodzi się (podobnie jak używany do tworzenia stron internetowych HTML) od języka SGML. SGML jest jednak stosunkowo niewygodny w użyciu, dlatego dopiero prostsza wersja, XML, została zaaprobowana przez społeczność informatyczną.

XML sam w sobie definiuje tylko sposób opisywania danych — konkretne znaczniki i metody ich obsługi zawierają standardy, które są tworzone na bazie XML-a. Jednym z najbardziej znanych jest XHTML, czyli HTML zgodny ze standardem XML. Inne popularne formaty to SVG i RSS. Ten ostatni zostanie omówiony dość dogłębnie, gdyż projekt opisywany w tym rozdziale będzie wykorzystywał ten właśnie format danych.

XML jest językiem bardzo ścisłym i rygorystycznym pod względem kontroli poprawności danych. Nie zamierzam opisać wszystkich zasad, którymi należy się kierować przy tworzeniu dokumentów XML; konkretne przykłady kodu zademonstruję przy okazji omawiania języka RSS.

Projekt — Czytnik RSS

Program, który utworzymy w tym rozdziale, będzie czytnikiem RSS. Aby lepiej zrozumieć ideę działania programu, omówię najpierw sam język RSS.

RSS służy do przesyłania nagłówków wiadomości. Dzięki temu w maksymalnie uproszczonej formie można pobierać najświeższe informacje z każdej witryny, która udostępnia jeden lub więcej *kanałów RSS*. Nagłówek wchodzący w skład kanału zawiera najczęściej tytuł, krótki opis oraz link do bardziej szczegółowych informacji. Wszystkie te dane zapisane są w postaci dokumentu XML. Jednym z przykładowych kanałów RSS jest lista bestsellerów udostępniana przez Wydawnictwo Helion — *http://helion.pl/rss/bestsellers.cgi*.

RSS występuje w kilku wersjach: 0.9x, 1.0 i 2.0. Nasz projekt będzie obsługiwać wersję 2.0.

Opis poszczególnych znaczników RSS znajduje się w dalszej części rozdziału.

J2ME a XML

Pośród dokumentów opisujących różnego rodzaju technologie związane z J2ME znajduje się JSR 172 *J2ME Web Services*, którego częścią jest specyfikacja parsera XML (*JAva Xml Parser — JAXP*).

Nasza aplikacja nie będzie jednak wykorzystywać tego API. Do napisania programu użyję zewnętrznej biblioteki o nazwie *kXML*. Powodem takiej decyzji jest niewielka dostępność implementacji JSR 172. Występuje ona w MIDP dopiero od wersji 2.0, i to nie we wszystkich modelach telefonów, które MIDP 2.0 obsługują. Tymczasem biblioteka kXML może być dołączona do każdego MIDletu — nawet działającego w MIDP 1.0 i CLDC 1.0. Dlaczego akurat w przypadku XML-a można zastosować taki zabieg, nie zwracając uwagi na wersję MIDP?

Otóż wiele dodatkowych API wymaga od telefonu specjalnej funkcjonalności. Do pełnej obsługi Mobile Media API, które jest tematem rozdziału 6., potrzebny jest w telefonie aparat fotograficzny. Bez niego Java nie jest w stanie wykonać zdjęć — co jest logiczne. Tymczasem obsługa XML-a to tak naprawdę szereg metod operujących na łańcuchach znaków. Zadaniem biblioteki kXML jest przetworzenie ciągu znaczników i wartości na bardziej przejrzyste i przydatne programiście struktury i zdarzenia.

kXML, w przeciwieństwie do mobilnego JAXP, umożliwia obsługę zarówno modelu SAX, jak i DOM. Modele te reprezentują sposoby parsowania dokumentu XML. Model SAX odpowiada za parsowanie pliku metodą „linijka po linijce". Informacje o kolejnych napotkanych znacznikach są przesyłane do programu w formie zdarzeń. Model ten potrzebuje do działania mniej pamięci, jest jednak zazwyczaj trudniejszy w obsłudze. Model DOM stanowi (przynajmniej pod tymi dwoma względami) przeciwieństwo modelu SAX. Program wczytuje cały dokument XML i przekształca go w strukturę drzewiastą. Dzięki temu programista może przechodzić swobodnie do wybranych gałęzi drzewa. Jest to oczywiście rozwiązanie prostsze, jednak wymaga zdecydowanie większej ilości pamięci.

Wykorzystanie biblioteki kXML w MIDletach

Aby móc wykorzystać dobrodziejstwa biblioteki kXML, należy pobrać ją, a następnie dodać do wybranego projektu J2ME. W naszym projekcie wykorzystamy bibliotekę kXML w wersji 1.21. Nie jest to najnowsza wersja, ale jest jedną z najczęściej wykorzystywanych spośród wszystkich wersji kXML.

Plik biblioteki znajduje się na płycie, w katalogu *biblioteka_XML*. Nosi on nazwę *kxml.jar*. Po utworzeniu projektu według standardowego schematu (*CzytnikRSS* — nazwa projektu) należy wykonać następujące kroki:

1. Utwórz w projekcie *CzytnikRSS* podkatalog *lib* — w Eclipse opcja *Folder* z menu *File->New*.
2. Skopiuj do podkatalogu *lib* plik *kxml.jar* z płyty (np. metodą *drag-and-drop*).
3. Otwórz okno właściwości projektu (kliknij prawym przyciskiem nazwę projektu w panelu *Package Explorer* i wybierz opcję *Properties*) i wybierz z listy opcję *Java Build Path*.
4. W zakładce *Libraries* kliknij przycisk *Add JARs*.
5. Z drzewa wybierz opcję *CzytnikRSS->lib->kxml.jar* i kliknij *OK*.
6. W oknie *Properties for CzytnikRSS* przejdź do zakładki *Order and Export* i zaznacz na liście *kxml.jar — CzytnikRSS/lib*. Kliknij *OK*.

Po wykonaniu powyższych operacji w panelu *Package Explorer* powinien pojawić się kolejny element o nazwie *kxml.jar*. Równocześnie uzyskamy dostęp do wszystkich funkcji tej biblioteki w kodzie MIDletu. Nasz projekt jest przygotowany do wprowadzenia kodu. Zaczniemy zatem od omówienia sposobu działania programu i interfejsu użytkownika.

Działanie programu i jego interfejs

Aby pobrać dokument RSS, należy znać jego URL. Dodatkowo, mając na uwadze koszta związane z pobieraniem danych, program powinien zaoferować użytkownikowi pobranie ograniczonej liczby nagłówków. Po pobraniu danych aplikacja wyświetla na liście tytuły wszystkich wiadomości. Przy użyciu menu użytkownik otrzymuje możliwość przejrzenia szczegółowych informacji na temat wybranej wiadomości. Nie można zapomnieć o kodowaniu znaków — każdy dokument XML zawiera informację o zastosowanym kodowaniu i nasz MIDlet powinien odczytywać je i ustawiać w obiekcie parsera.

Projekt zawiera cztery klasy, przeznaczone do:

- obsługi interfejsu użytkownika i MIDletu — *CzytnikRSSMIDlet.java*;
- pobrania pliku z internetu — *WatekPobierania.java*;
- analizy danych i przekształcania informacji w struktury wiadomości — *Parser.java*;
- reprezentowania struktury wiadomości — *Dane.java*.

Aplikacja składać się będzie z trzech wyświetlaczy. Pierwszy z nich tradycyjnie ma za zadanie pobrać od użytkownika dane potrzebne do wykonania kolejnych operacji. Drugi, zawierający spis wiadomości, zostanie utworzony z wykorzystaniem klasy `List`. Ostatnia formatka wyświetli szczegółowe informacje dotyczące wiadomości.

Tworzenie projektu zaczniemy, jak zwykle, od zadeklarowania zmiennych:

CzytnikRSSMIDlet.java

```
117: private Display ekran;
118: private Form formaWstepna;
119: private List lista;
120: private Form formaSzczegoly;
121: private TextField poleLiczba;
122: private TextField poleAdres;
123: private Parser parser;
124: private WatekPobierania watek;
125:}
```

Pola tekstowe służą do pobrania dwóch wartości wspomnianych w powyższym opisie. Formatka `formaWstepna` umożliwia pobranie danych, a `formaSzczegoly` — wyświetlenie pełnej informacji o danej wiadomości. Najważniejsze dla działania programu są obiekty innych klas zawartych w naszym pakiecie — czyli `parser` i `watek`. Następnym krokiem jest utworzenie interfejsu i przygotowanie MIDletu — w tym projekcie niezbyt rozbudowana czynność:

CzytnikRSSMIDlet.java

```
1: package czytnikrss;
2: import javax.microedition.lcdui.*;
3: import javax.microedition.midlet.MIDlet;
4: public class CzytnikRSSMIDlet extends MIDlet implements CommandListener
5: {
6:  public CzytnikRSSMIDlet()
7:  {
8:   formaWstepna = new Form("Czytnik RSS");
9:   poleLiczba = new TextField("Maks. liczba wiadomosci
     ↳(0-wszystkie):","0",5,TextField.NUMERIC);
10:  poleAdres = new TextField("Adres kanalu:","",120,TextField.ANY);
11:  formaWstepna.append(poleAdres);
12:  formaWstepna.append(poleLiczba);
13:  Command c = new Command("Wczytaj",Command.OK,0);
14:  formaWstepna.addCommand(c);
15:  formaWstepna.setCommandListener(this);
16:  lista = new List("Wybierz element",Choice.IMPLICIT);
17:  formaSzczegoly = new Form("Szczegoly");
18:  Command powrot = new Command("Powrot",Command.EXIT,0);
19:  formaSzczegoly.addCommand(powrot);
20:  formaSzczegoly.setCommandListener(this);
21:  ekran = Display.getDisplay(this);
22:  ekran.setCurrent(formaWstepna);
23:  parser = new Parser(this);
24: }
```

Lista wiadomości jest typu `IMPLICIT`, aby pozycje z menu były jak najłatwiejsze do wybrania. Najważniejsza instrukcja znajduje się w wierszu nr 23 — utworzenie obiektu parsera wymaga podania referencji do klasy MIDletu. Można z tego wywnioskować, że metody MIDletu będą wykorzystywane w klasie parsera. Następny fragment zawiera kilka metod pomocniczych. Z omówieniem pozostałych wstrzymam się do czasu, aż przybliżę zawartość podstawowej struktury danych niniejszej aplikacji — klasy `Dane`. Prawie wszystkie poniższe metody są wykorzystywane przez pozostałe klasy pakietu:

CzytnikRSSMIDlet.java

```
25: protected void startApp(){}
26: protected void pauseApp(){}
27: protected void destroyApp(boolean bezwarunkowo)
28: {
29:  if (watek!=null)
30:   watek.zakoncz();
31: }
32: public void usunElementy(Form formatka)
33: {
34:  for (int i=formatka.size()-1;i>=0;i--)
35:   formatka.delete(i);
36: }
...
85: public void aktywujListe()
86: {
87:  Command wybierz = new Command("Wybierz",Command.ITEM,0);
88:  Command info = new Command("Informacje",Command.ITEM,1);
89:  Command wyjscie = new Command("Koniec",Command.EXIT,0);
90:  lista.addCommand(wybierz);
91:  lista.addCommand(info);
92:  lista.addCommand(wyjscie);
93:  lista.setSelectCommand(wybierz);
94:  lista.setCommandListener(this);
95: }
96: public void pokazBlad(String s)
97: {
98:  Alert a = new Alert("Wystapil blad krytyczny! ");
99:  a.setString(s);
100:  a.setTimeout(3000);
101:  ekran.setCurrent(a);
102:  try
103:  {
104:   Thread.sleep(3000);
105:  } catch (InterruptedException e){}
106:  this.destroyApp(true);
107:  this.notifyDestroyed();
108: }
...
113: public int pobierzLiczbe()
114: {
115:  return Integer.parseInt(poleLiczba.getString());
116: }
```

Powyższy kod jest zbiorem różnych metod, z których każda zasługuje na odrębny opis. Metoda `destroyApp()` uwzględnia możliwość zakończenia działania aplikacji w trakcie pobierania dokumentu RSS. W takiej sytuacji program musi zakończyć połączenie

i zamknąć strumienie, aby nie zostawiać niezwolnionych zasobów. Kolejna metoda, `usunElementy()`, jest odpowiednikiem metody `deleteAll` zadeklarowanej w klasie `Form`. Niestety, metoda ta jest dostępna dopiero w MIDP 2.0. Dość nierozsądne byłoby zastosowanie MIDP 2.0 tylko ze względu na konieczność wykorzystania tej metody.

Metoda `usunElementy()` jest potrzebna, aby móc wielokrotnie wykorzystywać formatkę `formaSzczegoly` do wyświetlania szczegółowych informacji o wiadomości. Pętla jest wykonywana od największego indeksu do zera, ponieważ liczba znajdujących się na formatce elementów cały czas zmniejsza się (metoda `formatka.size()`), zatem odwrotne rozwiązanie w pewnym momencie doprowadziłoby do użycia indeksu spoza dopuszczalnego zakresu.

Po raz pierwszy w niniejszej książce zastosowane zostało rozwiązanie, w którym część procesu tworzenia interfejsu użytkownika przeniesiono poza konstruktor. Wynika to ze sposobu działania listy z wiadomościami. Otóż lista ta jest wyświetlana zaraz po nawiązaniu połączenia, jednak dodatkowe polecenia pojawiają się dopiero po pobraniu i wyświetleniu wszystkich wiadomości. Z tego względu tę część tworzenia interfejsu należy ulokować w osobnej metodzie.

Z metodą `pokazBlad()` spotkaliśmy się już w rozdziale 4. Teraz jednak instrukcja zatrzymująca działanie programu (`Thread.sleep()`) została wbudowana w treść metody.

Zadaniem ostatniej metody jest konwersja tekstowej zawartości pola na wartość liczbową — czynność, chociaż prosta, musi być opakowana w osobną metodę. Dzięki temu metoda może być wykorzystana w klasie `Parser`.

Zanim przejdę do opisu klasy `Dane` — struktury reprezentującej wiadomości — omówię język RSS, czyli esencję naszego projektu.

Język RSS

Nasz program wykorzystuje RSS w wersji 2.0. Oficjalna specyfikacja tej wersji znajduje się na stronie *http://blogs.law.harvard.edu/tech/rss*. MIDlet nie zawiera implementacji całego standardu (nieznane polecenia pomija), tak więc przy jego rozszerzaniu warto zapoznać się z treścią tej strony. Wykorzystywanie tej wersji nie oznacza, że program nie poradzi sobie z kanałami w starszych wersjach — zgodnie z przysłowiem diabeł tkwi w szczegółach, dlatego trzeba na początku określić, którą specyfikację uwzględnia nasz program.

Jak każdy dokument XML, tak i kanał RSS zawiera prolog, czyli informację o wersji zastosowanego XML-a i kodowaniu znaków. Informacja ta może wyglądać tak:

```
<?xml version="1.0" encoding="iso-8859-2"?>
```

Taka deklaracja oznacza, że dokument korzysta z XML-a w wersji 1.0 i kodowania znaków ISO środkowoeuropejskiego (jeden z najczęściej spotykanych, obok UTF-8, standardów kodowania dokumentów zawierających polskie znaki). Następną instrukcją może być `DOCTYPE`, jednak nie będę zajmował się nią szczegółowo. Pierwszym

znacznikiem, wskazującym na to, że dokument korzysta z języka RSS, jest znacznik <rss>. Należy umieścić w nim atrybut określający wersję RSS, jak poniżej:

```
<rss version="2.0">
```

Dokument RSS musi zawierać jeden kanał — opisany przy użyciu znacznika <channel>. Wewnątrz tego znacznika deklarowane są dwie podstawowe części: informacje o kanale i właściwa treść kanału, czyli wiadomości.

Informacje o kanale są reprezentowane przez pary znaczników znajdujących się bezpośrednio w znaczniku <channel>. Obowiązkowe znaczniki z tej grupy to:

- <title> — określa tytuł kanału;
- <link> — określa URL strony internetowej, która jest związana z kanałem;
- <description> — zawiera opis kanału.

Lista nieobowiązkowych znaczników jest znacznie dłuższa; poniżej znajdują się te najczęściej spotykane:

- <language> — określa język kanału. Język musi być określony według formatu zdefiniowanego w RFC 1766 (dostępny pod adresem *http://www.ietf.org/rfc/rfc1766.txt*). Najprostsza jego postać to kod dwuliterowy, np.: en (angielski), fr (francuski), pl (polski). Niektóre języki są jednak używane w wielu krajach. Aby określić w znaczniku kraj, należy dodać dwuliterowe oznaczenie kraju, np. en-us (angielski w Stanach Zjednoczonych Ameryki),
- <copyright> — określa notę dotyczącą praw autorskich dla kanału;
- <webMaster> — określa osobę odpowiedzialną za techniczną stronę kanału;
- <lastBuildDate> — zawiera datę ostatniej aktualizacji kanału. Data musi mieć format dzien_tygodnia dd miesiac rrrr hh:mm:ss, gdzie dzien_tygodnia i miesiac to trzyliterowe skróty odpowiednich nazw w języku angielskim, np. Jan — January — styczeń; Tue — Tuesday — wtorek;
- <category> — określa kategorię kanału;
- <image> — jest to znacznik złożony, określający obrazek związany z kanałem, np. jego logo. Zawiera trzy znaczniki:
  - <url> — określa adres obrazka (dopuszczalne formaty: JPG, PNG, GIF);
  - <title> — określa tytuł obrazka;
  - <link> — określa stronę, do której odwołuje się obrazek. Znaczniki <title> i <link> powinny mieć te same wartości co główne informacje kanału.

Po zdefiniowaniu powyższych znaczników można przystąpić do dodania konkretnych wiadomości do kanału. Pojedyncza wiadomość jest reprezentowana przez znacznik <item>. Zawiera on szereg znaczników, wśród których obowiązkowo musi się znaleźć przynajmniej tytuł **lub** treść. Poniższe zestawienie zawiera opis ważniejszych znaczników:

- <title> — zawiera tytuł wiadomości;
- <link> — zawiera URL strony internetowej zawierającej wiadomość;
- <description> — zawiera treść wiadomości;
- <author> — zawiera adres e-mail autora wiadomości (należy stosować, gdy kanał jest prowadzony przez wielu autorów);
- <category> — określa kategorię wiadomości;
- <guid> — określa unikalny identyfikator wiadomości; format tego identyfikatora nie jest precyzyjnie określony — może być to np. URL;
- <pubDate> — określa datę publikacji wiadomości;
- <source> — określa źródło, z którego pochodzi wiadomość. Jest ono określone w atrybucie `url`, np.: `<source url="http://serwis.org/newsy.xml">Gazeta Najnowsza</source>`.

Na zakończenie tego podrozdziału zaprezentuję treść prostego, przykładowego kanału (wszelkie zawarte w nim informacje są oczywiście zmyślone):

```
<?xml version="1.0" encoding="iso-8859-2"?>
<rss version="2.0">
 <channel>
  <title>Kanał - informacje o książce &quot;J2ME. Praktyczne projekty&quot;</title>
  <description>Kanał zawiera dużo informacji o książce omawiającej
  ↳różne zagadnienia związane z J2ME.</description>
  <link>http://helion.pl/autorzy/rykirz.htm</link>
  <category>Książki informatyczne</category>
  <webMaster>Krzysztof Rychlicki-Kicior</webMaster>
  <language>pl</language>
  <lastBuildDate>Wed, 21 Jun 2006 13:08:50 +0200</lastBuildDate>
  <item>
   <title>Nowa recenzja książki</title>
   <link>http://gazeta.informatyczna.org/recenzja.htm?tytul=j2me_pp</link>
   <description>Na stronie internetowej Gazety Maniaków Informatycznych pojawiła
   ↳się recenzja książki &quot;J2ME. Praktyczne projekty.&quot;. Zapraszamy
   ↳na strony Gazety.</description>
   <category>Recenzje</category>
  </item>
  <item>
   <title>Książka &quot;J2ME. Praktyczne projekty&quot; już na rynku!</title>
   <link>http://helion.pl/autorzy/rykirz.htm</link>
   <description>Tytułowa książka jest już dostępna na stronie internetowej
   ↳Wydawnictwa Helion. Zapraszamy do kupowania!</description>
   <category>Ważne informacje</category>
  </item>
 </channel>
</rss>
```

W powyższym dokumencie warto zwrócić uwagę na zapis `&quot;`. Jest to **encja** XML czyli symbol zastępujący pewien znak lub ciąg znaków. W tym przypadku oznacza ona po prostu cudzysłów — parser XML przetwarzając taki zapis zwróci pojedynczy znak ". Dodatkowo w dacie znalazła się konstrukcja `+0200` — oznacza ona strefę czasową, w której znajduje się kanał (według której jest aktualizowany).

Po zapoznaniu się z opisem języka RSS możemy przejść do omawiania sposobu reprezentowania struktur tego języka w naszym MIDlecie.

Struktura Dane

Klasa `Dane` służy do przechowywania:

- ogólnych informacjach na temat kanału (zgromadzonych w jednym obiekcie),
- szczegółów poszczególnych wiadomości (każda wiadomość — jeden obiekt).

Dane.java

```
43: private String[] dane;
44: private final String[] stalePl;
45: private final String[] staleOrg;
46:}
```

Główna zmienna klasy `Dane` to tablica `dane`. Zawiera ona wartości wszystkich znaczników z danego zakresu: albo główne informacje, albo składowe wiadomości. Znaczenie dwóch pozostałych, niezmiennych, tablic wyjaśnia się w konstruktorze:

Dane.java

```
1: package czytnikrss;
2: public class Dane
3: {
4:  public Dane(String[] org, String[] pl)
5:  {
6:   stalePl = pl;
7:   staleOrg = org;
8:   dane = new String[pl.length];
9:  }
```

W tablicy `staleOrg` są przechowywanie nazwy znaczników, które mają być pobierane z dokumentu RSS. Jeśli zatem chcemy pobrać tylko elementy `link`, `title` i `description`, to konieczne jest utworzenie 3-elementowej tablicy i przekazanie jej w parametrze `org`. Zadaniem tablicy `stalePl` jest przechowywanie polskich odpowiedników dla anglojęzycznych nazw znaczników (np. `title` — tytuł), zgodnie z indeksami w tablicy `staleOrg`. Będą one wyświetlane na formatce obok wartości składowych wiadomości lub informacji. Tablica `dane` musi mieć taką długość jak pozostałe tablice, dlatego wykorzystujemy długość jednej z nich w konstruktorze tablicy. Tablice `staleOrg` i `stalePl` posiadają modyfikator dostępu `final`, ponieważ ich wartości nie powinny być zmieniane w trakcie działania programu.

Kolejne pomocnicze metody nie są skomplikowane; ich głównym zadaniem jest udostępnianie zmiennych klasy i wykonywanie na nich prostych operacji:

Dane.java

```
10: public int sprawdz(String tekst)
11: {
12:  for (int i=0;i<staleOrg.length;i++)
```

```
13:  {
14:   if (staleOrg[i].equals(tekst))
15:    return i;
16:  }
17:  return -1;
18: }
19: public void ustawElement(String tekst, int indeks)
20: {
21:  dane[indeks] = tekst;
22: }
23: public String pobierzElement(String indeks)
24: {
25:  int k = this.sprawdz(indeks);
26:  if (k>-1)
27:   return this.pobierzElement(k);
28:  else
29:   return "";
30: }
31: public String pobierzElement(int indeks)
32: {
33:  return dane[indeks];
34: }
35: public String pobierzNazwe(int indeks)
36: {
37:  return stalePl[indeks];
38: }
39: public int rozmiar()
40: {
41:  return dane.length;
42: }
```

Najciekawsze metody to `sprawdz()` oraz `pobierzElement()`. Zadaniem pierwszej z nich jest sprawdzenie, czy w zestawie elementów określonych (w konstruktorze) dla danego obiektu znajduje się podany znacznik. Jeśli wyszukiwanie powiedzie się, zwracany jest indeks znalezionego elementu. Metoda ta jest często wykorzystywana w parserze danych oraz w metodzie `pobierzElement()`, która zwraca wartość elementu o podanej nazwie. Po uzyskaniu indeksu można wykorzystać prostszy wariant metody `pobierzElement()` (z parametrem typu `int`). Pozostałe metody zapewniają dostęp do poszczególnych zmiennych.

Obsługa poleceń w MIDlecie

Zapoznanie Czytelnika z klasą `Dane` było potrzebne, abym mógł bez problemu omówić działanie metody `commandAction()`:

CzytnikRSSMIDlet.java

```
37: public void commandAction(Command c, Displayable s)
38: {
39:  if (s == formaWstepna)
40:  {
41:   if (c.getCommandType() == Command.OK)
42:   {
43:    String url = poleAdres.getString();
```

```
44:    if (!url.equals(""))
45:    {
46:     ekran.setCurrent(lista);
47:     watek = new WatekPobierania(url, this, parser);
48:     watek.start();
49:    }
50:   }
51:  }
52:  if (s == lista)
53:  {
54:   if (c.getCommandType() == Command.ITEM)
55:   {
56:    this.usunElementy(formaSzczegoly);
57:    if (c.getPriority() == 0)
58:    {
59:     int k = lista.getSelectedIndex();
60:     Dane d = parser.pobierzElement(k);
61:     for (int i=0;i<d.rozmiar();i++)
62:      if (d.pobierzElement(i)!=null)
63:       formaSzczegoly.append(d.pobierzNazwe(i)+"\r\n"+d.pobierzElement(i)+
          ↳"\r\n");
64:    }
65:    if (c.getPriority() == 1)
66:    {
67:     Dane daneKanalu = parser.pobierzDaneKanalu();
68:     for (int i=0;i<daneKanalu.rozmiar();i++)
69:      if (daneKanalu.pobierzElement(i)!=null)
70:       formaSzczegoly.append(daneKanalu.pobierzNazwe(i)+"\r\n"+
          ↳daneKanalu.pobierzElement(i)+"\r\n");
71:    }
72:    ekran.setCurrent(formaSzczegoly);
73:   }
74:   if (c.getCommandType() == Command.EXIT)
75:   {
76:    this.destroyApp(true);
77:    this.notifyDestroyed();
78:   }
79:  }
80:  if (s == formaSzczegoly)
81:  {
82:   ekran.setCurrent(lista);
83:  }
84: }
...
109: public void dodajElement(Dane d)
110: {
111:  lista.append(d.pobierzElement("title"),null);
112: }
```

Pierwsza część kodu (wiersze 41 – 51) obejmuje proces pobierania dokumentu XML z internetu. Do tego celu wykorzystywana jest klasa `WatekPobierania`, która jest uruchamiana w osobnym wątku — pobieranie danych może trwać dość długo, nawet gdy rozmiar dokumentu jest niewielki. Najważniejsze są zdarzenia listy wiadomości. Pierwsze z nich służy do wyświetlenia szczegółów wybranej wiadomości, drugie — do wyświetlenia głównych informacji o kanale. Niezależnie od rodzaju zdarzenia formatka `formaSzczegoly` musi być wyczyszczona z uprzednio dodanych etykiet. Po uzyskaniu

indeksu wiadomości jest ona pobierana za pomocą parsera. Następnie program sprawdza, czy w wiadomości zostały zawarte kolejne znaczniki zadeklarowane w programie (ich lista znajduje się w klasie `Parser`). Jeśli kanał RSS nie zawiera jakiegoś znacznika (sprawdzenie odbywa się w wierszu nr 62), program przechodzi dalej; w przeciwnym razie do formatki zostaje dodana etykieta. Wyświetlanie głównych informacji o kanale to podobny proces; różni się sposobem uzyskania danych w parserze. Zestaw pól dotyczących informacji o kanale zazwyczaj różni się od zestawu pól dla wiadomości (chociaż część znaczników jest identyczna).

Druga metoda zawarta w powyższym listingu jest wykorzystywana przez klasę `Parser` do utworzenia listy wiadomości. W tym celu pobierany jest znacznik `title` — tytuł wiadomości.

Pobieranie dokumentu RSS

Pierwszą czynnością, którą wykonuje MIDlet po wprowadzeniu danych kanału, jest pobranie jego zawartości. Do tego celu służy wspomniana klasa `WatekPobierania`. Rozszerza ona klasę `Thread` i zawiera w metodzie `run()` instrukcje związane z nawiązaniem połączenia internetowego.

WatekPobierania.java

```
38: private HttpConnection pol;
39: private CzytnikRSSMIDlet midlet;
40: private String url;
41: private InputStream in;
42: private Parser parser;
43:}
```

Zmienne `url` i `pol` służą do nawiązania połączenia, a strumień `in` udostępnia parserowi dokument RSS. Zarówno obiekt MIDletu, jak i parsera musi być dostępny w tej klasie, gdyż klasach metody są wykorzystywane w metodzie `run()` wątku:

WatekPobierania.java

```
1: package czytnikrss;
2: import javax.microedition.io.*;
3: import java.io.*;
4: public class WatekPobierania extends Thread
5: {
6:  public WatekPobierania(String url, CzytnikRSSMIDlet midlet, Parser parser)
7:  {
8:   this.url = url;
9:   this.midlet = midlet;
10:  this.parser = parser;
11: }
12: public void run()
13: {
14:  try
15:  {
```

```
16:   pol = (HttpConnection)Connector.open(url);
17:   in = pol.openInputStream();
18:   parser.ustawStrumien(in, midlet.pobierzLiczbe());
19:  }
20:  catch (IOException e)
21:  {
22:   e.printStackTrace();
23:   this.zakoncz();
24:   midlet.pokazBlad(e.getMessage());
25:   return;
26:  }
27: }
```

Klasa nawiązuje połączenie w znany sposób. Następnie parser otrzymuje obiekt strumienia wraz z liczbą wiadomości, które ma pobrać — są to jedyne dwie zmienne, których potrzebuje do działania. W przypadku błędu podczas komunikacji program musi zamknąć połączenie i zakończyć działanie. Metoda, która gwarantuje zwolnienie zasobów nie tylko w przypadku błędu, ale również normalnego zakończenia działania programu (przez metodę `destroyApp()` MIDletu), nazywa się `zakoncz()`:

WatekPobierania.java

```
28: public void zakoncz()
29: {
30:  try
31:  {
32:   if (pol!=null)
33:    pol.close();
34:   if (in!=null)
35:    in.close();
36:  } catch (Exception e){}
37: }
```

Zamykane jest w niej zarówno połączenie jak i strumienia wejściowy.

Jak zostało to pokazane w metodzie `run()`, w wątku pobierania zostaje wywołana metoda parsera. Jej zadaniem jest przeanalizowanie pobranej treści i wyłuskanie z niej interesujących nas danych. Zanim jednak omówię odpowiedzialną za to klasę, czyli `Parser`, warto zapoznać się z biblioteką *kXML*.

Piękne jak gra na SAXofonie — biblioteka kXML pod lupą

W naszym programie do analizy dokumentu XML wykorzystywany jest model SAX. To dlatego, że model ten wymaga mniejszej ilości pamięci i jest bardziej wydajny; ponadto korzystanie z modelu DOM wymaga odczytania całego dokumentu, co zwiększa ilość danych pobieranych z internetu.

Po dodaniu biblioteki *kXML* do projektu można korzystać w nim z następujących pakietów:

- `org.kxml` — zawiera podstawowe, wspólne elementy wszystkich klas z całej biblioteki;
- `org.kxml.io` — zawiera klasy przeznaczone do zapisu danych w formacie XML;
- `org.kxml.kdom` (opcjonalny) — zawiera klasy przeznaczone do obsługi modelu DOM;
- `org.kxml.parser` — zawiera klasy przeznaczone do obsługi modelu SAX;
- `org.kxml.wap` — zawiera klasy do obsługi WML — protokołu będącego odpowiednikiem HTTP dla telefonów komórkowych i urządzeń mobilnych.

Najbardziej interesujący nas pakiet to `org.kxml.parser`. Znajdują się w nim dwie klasy, które w naszym projekcie pełnią kluczową rolę:

- `XmlParser` — służy do wczytywania i analizy danych z pliku XML;
- `ParseEvent` — reprezentuje pojedyncze zdarzenie odpowiadające elementowi dokumentu XML.

Wczytywanie i analiza danych

Obiekt klasy `XmlParser` wczytuje kolejne elementy dokumentu XML i zwraca je w postaci obiektów klasy `ParseEvent`. Można z nich odczytać nazwy, atrybuty i typy elementów. Zwłaszcza ta ostatnia właściwość będzie często wykorzystywana w projekcie. Początkujący programiści wykorzystujący w swoich programach język XML często zapominają, że elementem jest nie tylko znacznik, ale i wartość tekstowa. Co za tym idzie, znaki końca wiersza, którymi przedzielane są często znaczniki, to również elementy dokumentu. Należy o tym pamiętać przy analizie kolejnych zdarzeń SAX. Przykładowy kod tworzący obiekt parsera może wyglądać tak:

```
HttpConnection pol = (HttpConnection)Connector.open(adresURL);
InputStream is = pol.openInputStream();
XmlParser parser = new XmlParser(new InputStreamReader(is, "ISO-8859-1"));
```

Ponieważ konstruktor klasy `XmlParser` wymaga podania obiektu czytnika, a nie strumienia, można wykorzystać dowolne kodowanie i poprawnie odczytać np. polskie znaki.

Należy pamiętać, że w przeciwieństwie do emulatora prawdziwe urządzenia nie udostępniają zbyt dużej liczby kodowań. Sposób obsługi sytuacji, gdy telefon nie obsługuje żądanego kodowania, omówię w jednym z następnych podrozdziałów.

Po utworzeniu obiektu parsera można przystąpić do wczytywania dokumentu XML. Wykorzystuje się do tego celu metodę `read()`, która zwraca obiekt klasy `ParseEvent` i jednocześnie przesuwa wskaźnik pozycji w dokumencie. Jeśli istnieje potrzeba „podejrzenia" znacznika bez przesuwania wskaźnika, należy zastosować metodę `peek()`.

Trudno określić jeden schemat wczytywania danych; można jednak wyróżnić pewne wspólne cechy każdego algorytmu, który wczytuje dokument XML korzystając z modelu SAX (za pomocą biblioteki *kXML*):

- zawiera pętlę `while`; warunkiem jej zakończenia jest to, aby pobrany element miał typ `END_DOCUMENT`;
- w każdej iteracji pętli pojawia się wywołanie metody `read()`.

Jak już wspomniałem wcześniej, istotną rolę podczas analizy odgrywa typ elementu. Najważniejsze typy, które obsługuje parser *kXML*, to:

- `END_DOCUMENT` — koniec dokumentu;
- `START_TAG` — znacznik otwierający;
- `END_TAG` — znacznik zamykający;
- `TEXT` — wartość tekstowa;
- `WHITESPACE` — wartość tekstowa złożoną ze spacji i innych znaków odstępu.

Wszystkie powyższe stałe są zadeklarowane w klasie `org.kxml.Xml`. Gdy dysponujemy obiektem klasy `ParseEvent`, możemy wykonać różne metody dające dostęp do nazwy, atrybutów i typu danego elementu. Jedną z zalet *kXML* jest zachowanie tych metod w sytuacji, gdy program wywołuje je w nieodpowiednim momencie. Wiadomo, że element tekstowy nie ma atrybutów, a znacznik nie ma wartości tekstowej. Odwołanie do metod nieodpowiednich dla danego typu elementu nie powoduje jednak wygenerowania wyjątku, jak w przypadku niektórych parserów, tylko zwraca wartości `null`.

`org.kxml.parser.XmlParser`

`public XmlParser(Reader czytnik)` — tworzy obiekt parsera XML według modelu SAX; wczytuje dane, korzystając z danego czytnika.

`public ParseEvent read()` — wczytuje następny element dokumentu XML.

`public ParseEvent peek()` — wczytuje następny element bez przesuwania wskaźnika pozycji w dokumencie.

`org.kxml.parser.ParseEvent`

`public int getType()` — zwraca typ elementu zgodnie ze stałymi opisanymi w tym podrozdziale.

`public String getText()` — zwraca wartość tekstową elementu, o ile jest on typu `TEXT`, `PROCESSING_INSTRUCTION` lub `DOCTYPE`.

`public Vector getAttributes()` — zwraca atrybuty elementu w postaci wektora (o ile element jest typu `START_TAG`).

`public String getValue(String atrybut)` — zwraca wartość danego atrybutu (o ile element jest typu `START_TAG`).

Parser w praktyce

Po zakończeniu pobierania pliku program zaczyna aktywnie korzystać z klasy `Parser`. Jest to najbardziej skomplikowana klasa z całego programu, jednak po wcześniejszym omówieniu biblioteki *kXML* i reszty programu nie powinno być problemu ze zrozumieniem poniższego kodu.

Zadaniem klasy `Parser` jest udostępnienie w postaci obiektów klasy `Dane` głównych informacji o kanale oraz poszczególnych wiadomości. To właśnie w klasie `Parser` są zadeklarowane stałe określające nazwy analizowanych i wczytywanych znaczników:

Parser.java

```
129: private CzytnikRSSMIDlet m;
130: private InputStream wej;
131: private Vector elementy;
132: private Dane daneKanalu;
133: public static final String[] STALE_KANALU_ORG = new String[]
134:    {"title", "link", "description","language","pubDate"};
135: public static final String[] STALE_KANALU_PL = new String[]
136:    {"Tytul:","Adres:","Opis:","Jezyk:","Data publikacji:"};
137: public static final String[] STALE_ELEMENTU_ORG = new String[]
138:    {"title", "link", "description","pubDate","author","category"};
139: public static final String[] STALE_ELEMENTU_PL = new String[]
140:    {"Tytul:","Adres:","Opis","Data:","Autor:","Kategoria:"};
141:}
```

Dwie pierwsze zmienne są przekazywane w metodzie `run()` klasy `WatekPobierania`. Zwłaszcza strumień pełni ważną rolę — jest źródłem danych dla parsera. Kolejne dwie zmienne przechowują wynik działania parsera. Wektor `elementy` przechowuje listę wiadomości (złożoną z obiektów klasy `Dane`), a główne informacje znajdują się w pojedynczym obiekcie `daneKanalu`.

W deklaracji stałych odnajdujemy tajemnicze listy znaczników. Stałe te zawierają (w kolejności):

- listę nazw znaczników pobieranych z głównych informacji o kanale,
- polskie nazwy używane przy wyświetlaniu informacji o kanale,
- listę nazw znaczników pobieranych z wiadomości (w znaczniku `<item>`),
- polskie nazwy używane przy wyświetlaniu pełnej treści wiadomości.

Konstruktor klasy pobiera obiekt MIDletu. Parser potrzebuje go do wywołania metod wyświetlających informacje na formatce `formaSzczegoly`.

Parser.java

```
1: package czytnikrss;
2: import java.io.*;
3: import java.util.Vector;
4: import org.kxml.parser.XmlParser;
5: import org.kxml.parser.ParseEvent;
```

```
6: import org.kxml.Xml;
7: public class Parser
8: {
9:  public Parser(CzytnikRSSMIDlet m)
10: {
11:  this.m = m;
12: }
```

Z biblioteki *kXML* wykorzystujemy tylko trzy omówione w poprzednim podrozdziale klasy.

Jedyną metodą wywoływaną spoza klasy jest metoda `ustawStrumien()`. Ustawia ona strumień wejścia i rozpoczyna proces parsowania:

Parser.java

```
120: public void ustawStrumien(InputStream in, int liczba)
121: {
122:  wej = in;
123:  this.parsuj(liczba);
124: }
```

Metoda ta jest wywoływana w metodzie `run()` klasy `WatekPobierania`. Możemy teraz przejść do istoty całego programu — metody `parsuj()`:

Parser.java

```
77: public void parsuj(int liczba)
78: {
79:  if (wej==null) return;
80:  try
81:  {
82:   String s = this.zwrocProlog();
83:   XmlParser parser = this.utworzParser(s);
84:   ParseEvent t;
85:   daneKanalu = new Dane(STALE_KANALU_ORG, STALE_KANALU_PL);
86:   elementy = new Vector();
87:   do
88:   {
89:    t = parser.read();
90:    this.analizuj(parser, t, daneKanalu);
91:   }
92:   while (t.getType() != Xml.END_DOCUMENT &&
      ↳((t.getType()==Xml.START_TAG)?!t.getName().equals("item"):true));
      ↳// koniec cz. 1.
```

Powyższa metoda wykorzystuje wiele pomocniczych metod; każda z nich jest dokładnie opisana w dalszej części podrozdziału. Wiersze 82 i 83 pojawiają się w kodzie z powodu jednej z nielicznych niedoskonałości biblioteki *kXML*. Otóż parser całkowicie pomija prolog (nie są generowane żadne związane z nim zdarzenia) — po prostu wczytuje dane, aż napotka pierwszy znacznik właściwej treści. Na szczęście dysponujemy strumieniem wejściowym — dzięki temu możemy wczytać prolog ręcznie. Odpowiada za to metoda `zwrocProlog()` — wczytuje cały prolog i przesuwa wskaźnik pozycji w strumieniu. Nie wpływa to na pracę parsera, gdyż do momentu napotkania pierwszego znacznika nie analizuje on wczytywanych danych. Metoda `zwrocParser()` wykorzystuje prolog, aby

pobrać z niego kodowanie użyte w dokumencie XML. Na podstawie strumienia i kodowania jest tworzony obiekt parsera, który jest wykorzystywany w metodzie `parsuj()`. Po utworzeniu obiektów przechowujących dane wynikowe rozpoczyna się pętla, w której wczytywane są kolejne znaczniki. Metoda `analizuj()` zajmuje się dokładnym sprawdzeniem zawartości znacznika — na razie wystarczy powiedzieć, że w pętli są pobierane kolejne główne informacje.

Bardzo ważny jest warunek zakończenia wczytywania głównych informacji. Pierwsza część warunku to napotkanie końca dokumentu — w praktyce niespotykane, ale dobrze jest być przygotowanym na każdą ewentualność. Drugi warunek jest zależny od rodzaju wczytanego elementu. Jeśli jest to tekst lub np. znacznik zamykający, pętla może być kontynuowana. Jeśli jednak mamy do czynienia ze znacznikiem otwierającym, trzeba sprawdzić, czy nie jest to znacznik `<item>` — a zarazem pierwsza wiadomość. W tej sytuacji trzeba zakończyć wczytywanie ogólnych informacji i przejść do odczytu listy wiadomości.

Parser.java

```
94:   int licznik = 0;
95:   while ((t.getType() != Xml.END_DOCUMENT)&&((liczba==0)?true:licznik<liczba))
96:   {
97:    if (t.getType()==Xml.START_TAG && t.getName().equals("item"))
98:    {
99:     Dane de = new Dane(STALE_ELEMENTU_ORG,STALE_ELEMENTU_PL);
100:     do
101:     {
102:      this.analizuj(parser, t, de);
103:      t = parser.read();
104:     } while ((t.getType()==Xml.END_TAG)?!t.getName().equals("item"):true);
105:     elementy.addElement(de);
106:     licznik += 1;
107:    }
108:    t = parser.read();
109:   }
110:  }
111:  catch (Exception e)
112:  {
113:   e.printStackTrace();
114:   m.pokazBlad(e.getMessage());
115:  }
116:  for (int i=0;i<elementy.size();i++)
117:   m.dodajElement((Dane)elementy.elementAt(i));
118:  m.aktywujListe();
119: }
```

Druga pętla w tej metodzie również ma dość złożony warunek. Tradycyjnie jest sprawdzane wystąpienie końca dokumentu; drugą część stanowi jednak sprawdzenie liczby wczytanych wiadomości. Przy liczbie równej 0 wczytywane są wszystkie wiadomości (stąd wartość `true` w operatorze trójargumentowym), natomiast dla liczby określonej przez użytkownika stosowany jest licznik.

W wierszu nr 97 sprawdzany jest warunek oznaczający początek analizy znacznika. Musi być to otwierający znacznik `<item>`. Następnie wewnątrz tego znacznika wykonywany jest proces bardzo podobny do pobierania głównych informacji o kanale (wiersze

100 – 104). Różnica polega oczywiście na warunku zakończenia pętli. Składowe wiadomości są dodawane do obiektu `de`, dopóki parser nie napotka znacznika zamykającego `</item>`. Następnie gotowy element jest dodawany do wektora `elementy`.

Po wczytaniu całego dokumentu parser wyświetla wszystkie nagłówki wiadomości (wiersze 116 – 117) i dodaje polecenia do listy za pomocą metody `aktywujListe()` klasy MIDletu.

Omówienie metod pomocniczych rozpocznę od `zwrocProlog()` i `utworzParser()`, zastosowanych w analizowanym wcześniej kodzie:

Parser.java

```
29: private String zwrocProlog() throws Exception
30: {
31:  int b = 0;
32:  StringBuffer bufor = new StringBuffer();
33:  while (b!=-1 && ((char)b) != '>')
34:  {
35:   b = wej.read();
36:   bufor.append((char)b);
37:  }
38:  if (b==-1) throw new Exception("Nie znaleziono pliku RSS");
39:  return bufor.toString();
40: }
```

Wczytywanie prologu odbywa się do momentu napotkania znaku > zamykającego znacznik lub nagłego przerwania wczytywania pliku. W tym drugim przypadku metoda zwraca wyjątek. Zawartość bufora jest konwertowana na łańcuch znaków; stanowi on źródło danych dla metody `utworzParser()`:

Parser.java

```
41: public XmlParser utworzParser(String s) throws Exception
42: {
43:  XmlParser parser;
44:  final String KODOWANIE = "encoding=";
45:  int j = s.indexOf(KODOWANIE);
46:  if (j>-1)
47:  {
48:   s = s.substring(j);
49:   int l = s.indexOf('"',KODOWANIE.length()+1);
50:   String wynik = s.substring(KODOWANIE.length()+1,l);
51:   if (!wynik.equals(""))
52:   try
53:   {
54:    parser = new XmlParser(new InputStreamReader(wej,wynik));
55:   } catch (Exception e)
56:   {
57:    parser = new XmlParser(new InputStreamReader(wej));
58:   }
59:   else
60:    parser = new XmlParser(new InputStreamReader(wej));
61:  } else
62:   parser = new XmlParser(new InputStreamReader(wej));
63:  return parser;
64: }
```

Powyższą metodę można podzielić na dwie części. Na początku z łańcucha o postaci:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

trzeba wyłuskać fragment:

```
ISO-8859-2
```

czyli pobrać wartość atrybutu `encoding`. Kod działa następująco (w nawiasie efekt po wykonaniu danej operacji dla prologu podanego powyżej):

- pobiera indeks, od którego zaczyna się ciąg znaków `encoding=`, a następnie pobiera część łańcucha od tego indeksu do końca (efekt: `encoding="ISO-8859-2"?>`),
- pobiera indeks cudzysłowu zamykającego nazwę kodowania i kopiuje łańcuch od pierwszego znaku za pierwszym cudzysłowem do pierwszego znaku przed drugim cudzysłowem (efekt: `ISO-8859-2`).

Druga część metody jest odpowiedzialna za utworzenie obiektu parsera. Jedyną problematyczną kwestią jest zastosowanie kodowania znaków. W przypadku niemożności utworzenia parsera z podanym w dokumencie XML kodowaniem, metoda utworzy parser z kodowaniem domyślnym.

Kolejną bardzo ważną metodą jest `analizuj()`:

Parser.java

```
65: public void analizuj(XmlParser parser, ParseEvent t, Dane d) throws IOException
66: {
67:  if (t.getType() == Xml.START_TAG)
68:  {
69:   int k = d.sprawdz(t.getName());
70:   if (k>-1)
71:   {
72:    String wartosc = this.pobierz(parser);
73:    d.ustawElement(wartosc, k);
74:   }
75:  }
76: }
```

Metoda ta otrzymuje fragment kodu o postaci takiej jak poniższa:

```
<title>Bardzo fajny tytul</title>
```

Jej zadaniem jest sprawdzenie, czy określony znacznik (w tym przypadku `title`) występuje w podanym jako parametr obiekcie klasy `Dane` (wywołanie metody `sprawdz()` — wiersze nr 69). Jeśli tak (wiersze nr 70), metoda `pobierz()` wydziela wartość znacznika (`Bardzo fajny tytul`) i ustawia ją na określonej pozycji w obiekcie `d`:

Parser.java

```
13: public String pobierz(XmlParser parser) throws IOException
14: {
15:  ParseEvent t;
16:  String wartosc = "";
17:  do
```

```
18:  {
19:   t = parser.read();
20:   if (t.getType() == Xml.TEXT)
21:    wartosc += t.getText();
22:  } while (t.getType()!=Xml.END_TAG);
23:  return wartosc;
24: }
```

W momencie uruchomienia tej metody wskaźnik pozycji w dokumencie jest ustawiony tuż za znacznikiem otwierającym, <title>. Metoda wczytuje kolejne elementy i dodaje ich zawartość do zwracanej wartości. Proszę zwrócić uwagę, że jeśli wewnątrz danego znacznika znajdą się dwa elementy typu TEXT, wtedy zwrócone zostanie połączenie tych dwóch elementów. Jest to jednak sytuacja rzadka. Warunkiem zakończenia pętli jest napotkanie znacznika zamykającego (w naszym przypadku — </title>).

Klasa Parser zawiera jeszcze dwie metody:

Parser.java

```
25: public Dane pobierzDaneKanalu()
26: {
27:  return daneKanalu;
28: }
...
125: public Dane pobierzElement(int indeks)
126: {
127:  return (Dane)elementy.elementAt(indeks);
128: }
```

Metody te udostępniają zmienne daneKanalu i elementy. W przypadku pobierania elementu trzeba posłużyć się indeksem i dokonać rzutowania na typ Dane (niestety w J2ME nie można używać klas generycznych).

Efekt działania programu widać na rysunkach 5.1 i 5.2.

Rysunek 5.1. *Wyświetlanie listy elementów*

Podsumowanie

Czytnik RSS to jeden z ciekawszych projektów. Może być uruchamiany na platformie MIDP 1.0 i CLDC 1.0. Wykorzystuje zewnętrzną bibliotekę i łączy w sobie wiele najnowszych technologii: XML, strumienie, internet.

Rysunek 5.2.
Wyświetlanie szczegółów elementu

Możliwości rozbudowy tego projektu nie są bynajmniej małe. Można dodać opcję archiwizacji pobranych wiadomości — za pomocą RMS. Należy też pamiętać, że program obsługuje tylko część formatu RSS — dodanie obsługi pozostałych jego elementów, zwłaszcza elementu `image`, może być bardzo ciekawym wyzwaniem.