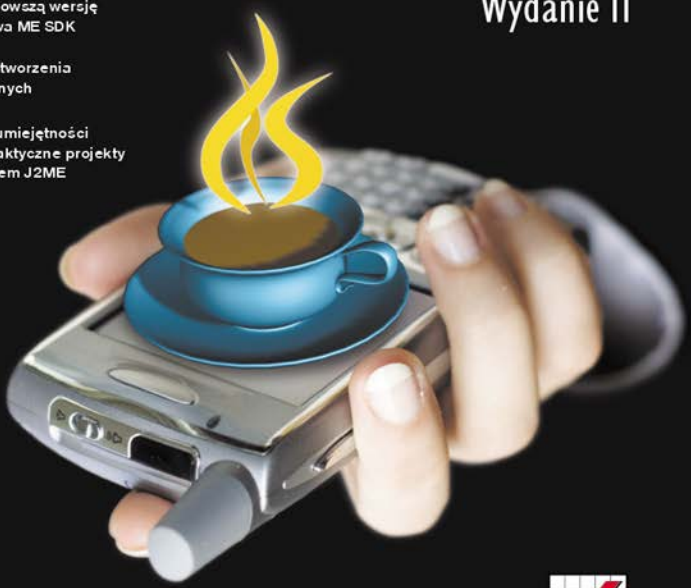
Naucz się efektywnie wykorzystywać
możliwości oferowane przez J2ME!

J2ME

Praktyczne projekty

Wydanie II

- Zainstaluj najnowszą wersję środowiska Java ME SDK
- Poznaj zasady tworzenia aplikacji mobilnych
- Rozwiń swoje umiejętności w oparciu o praktyczne projekty z wykorzystaniem J2ME



Krzysztof Rychlicki-Kicior

Helion

J2ME. Praktyczne projekty. Wydanie II

Autor: Krzysztof Rychlicki-Kicior
ISBN: 978-83-246-2835-3
Format: 158×235, stron: 272

Naucz się efektywnie wykorzystywać możliwości oferowane przez J2ME!

- Zainstaluj najnowszą wersję środowiska Java ME SDK
- Poznaj zasady tworzenia aplikacji mobilnych
- Rozwiń swoje umiejętności w oparciu o praktyczne projekty z wykorzystaniem J2ME

J2ME, czyli Java 2 Micro Edition, to uproszczona wersja platformy Java, opracowana przez firmę Sun Microsystems specjalnie dla potrzeb programowania urządzeń przenośnych, takich jak telefony komórkowe czy palmtopy. Umożliwia tworzenie ciekawych i wydajnych aplikacji mobilnych, które bez większych problemów można uruchamiać na sprzęcie o stosunkowo słabych parametrach technicznych. Pozwala to osobom zainteresowanym produkcją gier, programów multimedialnych czy narzędzi sieciowych swobodnie rozwinąć skrzydła w tej dziedzinie.

„J2ME. Praktyczne projekty. Wydanie II" to przydatny przewodnik po zaawansowanych zagadnieniach, związanych z tworzeniem różnego rodzaju aplikacji mobilnych przy użyciu środowiska Java. Autor pokrótce przedstawia w nim podstawowe informacje na temat projektowania i kodowania programów działających na urządzeniach przenośnych, aby szybko przejść do konkretnych przykładów zastosowania zdobytej wiedzy. Dzięki nim nauczysz się tworzyć gry, aplikacje komunikacyjne, programy multimedialne i narzędzia GPS. Jeśli chcesz szybko opanować J2ME, tej książki nie może zabraknąć na Twojej półce!

- Instalacja środowiska programisty J2ME
- Podstawowe informacje o platformie i sposobach jej używania
- Obsługa zaawansowanych wyświetlaczy
- Tworzenie aplikacji sieciowych i komunikacyjnych
- Przetwarzanie i wykorzystywanie danych XML
- Tworzenie aplikacji multimedialnych i obsługa kamer
- Projektowanie i programowanie gier
- Tworzenie aplikacji GPS

Dołącz do elitarnego grona programistów aplikacji mobilnych!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 6.

Multimedia w Twoim telefonie

Telefony komórkowe coraz częściej zawierają dodatkowe udogodnienia, dzięki którym przestają być tylko urządzeniami do prowadzenia rozmów. Jednym z najpopularniejszych sposobów przyciągania uwagi klientów jest dodawanie możliwości multimedialnych, takich jak:

- odtwarzanie plików dźwiękowych (midi/wav/mp3),
- odtwarzanie plików wideo (mpeg),
- nagrywanie audio,
- wykonywanie zdjęć,
- nagrywanie plików wideo.

J2ME umożliwia programistom wykorzystanie tych możliwości, o ile dany telefon obsługuje sprzętowo daną funkcję.

Projekt — odtwarzacz multimedialny

Jak sama nazwa wskazuje, w niniejszym projekcie zostaną zaprezentowane wybrane możliwości multimedialne, które oferuje J2ME. Równie ważną cechą programu będzie obsługa wielu rodzajów źródeł danych — odtwarzacz będzie mógł wczytywać dane z internetu (za pomocą protokołu HTTP), z lokalnego systemu plików (za pomocą klas z pakietu `javax.microedition.io.file`) oraz z RMS. Oprócz odtwarzania zasobów multimedialnych program umożliwi nagranie dźwięku i zrobienie zdjęcia — z możliwością zapisu w RMS.

Na początku warto dowiedzieć się, gdzie są zadeklarowane klasy obsługujące multimedia i jakie warunki musi spełniać urządzenie, aby dany rodzaj multimediów odtworzyć.

Obsługa multimediów w telefonach

Najbardziej funkcjonalnym API (w momencie pisania tej książki — cały czas są tworzone jego kolejne wersje) jest Mobile Media API 1.1, zdefiniowane w JSR-135. Zawiera ono klasy i interfejsy przeznaczone do wykonywania wszystkich wspomnianych na początku rozdziału czynności. Jak to zwykle bywa, API to nie jest powszechnie dostępne w telefonach komórkowych. Dlatego najbardziej podstawowe elementy zawarte w MMAPI zostały wydzielone i weszły w skład MIDP 2.0. Owa część nosi nazwę MIDP 2.0 Media API i jedynym warunkiem, który musi spełniać telefon, aby możliwe była jego wykorzystanie, jest dostępność MIDP w wersji 2.0.

Za obsługę multimediów odpowiadają pakiety `javax.microedition.media`, `javax.microedition.media.control` oraz `javax.microedition.media.protocol`. Zawartość tych pakietów można podzielić na trzy części:

- elementy dostępu do danych — na podstawie URL udostępniają strumienie danych;
- odtwarzacze — kontrolują proces przygotowania i odtwarzania (a także nagrywania) danych multimedialnych;
- kontrolki — odpowiadają za konkretną właściwość odtwarzacza, np. głośność lub wysokość dźwięku.

Różnice między MMAPI a MIDP Media API

Jak wspomniałem, MIDP Media API jest podzbiorem MMAPI. Poniższe zestawienie zawiera zbiór wszystkich elementów dostępnych w MIDP Media API:

- klasy: `Manager`;
- klasy wyjątków: `MediaException`;
- interfejsy: `Control`, `Controllable`, `Player`, `PlayerListener`, `ToneControl`, `VolumeControl`.

MMAPI zawiera wszystkie powyższe elementy oraz szereg innych. Spora część z nich zostanie wykorzystana w projekcie.

Proces odtwarzania plikulnych

W niniejszym podrozdziale opiszę wszystkie czynności, które trzeba wykonać, aby odtworzyć plik multimedialny. Program zazwyczaj dysponuje adresem internetowym lub ścieżką dostępu do pliku. Na początek należy więc zapoznać się z metodą `createPlayer()` klasy `Manager`. Tworzy ona obiekt odtwarzacza (klasy `Player`) na podstawie podanego adresu lub strumienia:

```
Player odtwarzacz = Manager.createPlayer("http://serwer.org/plik.wav");
```

Dysponując gotowym obiektem odtwarzacza, należy wspomnieć o **stanach**, w jakich może się on znajdować. Są one określone następującymi stałymi (w kolejności od stanu początkowego do odtwarzania):

- `UNREALIZED` — odtwarzacz jest tuż po utworzeniu. Nie można wykonać większości jego metod.
- `REALIZED` — odtwarzacz ma informacje potrzebne do pobrania danych.
- `PREFETCHED` — odtwarzacz dysponuje pobranymi danymi; jest gotowy do rozpoczęcia odtwarzania.
- `STARTED` — odtwarzacz jest w trakcie odtwarzania pliku multimedialnego. W przypadku przerwania odtwarzania przechodzi z powrotem do stanu `PREFETCHED`.
- `CLOSED` — odtwarzacz kończy działanie i zwalnia zaalokowane zasoby.

Aby przejść do danego stanu, należy wywołać metody: `realize()`, `prefetch()`, `start()`, `close()`. Metoda `stop()` przerywa działanie odtwarzacza i powoduje przejście do stanu `PREFETCHED`.

`javax.microedition.media.Manager`

- `public Player createPlayer(String` *`url`*`)` — tworzy obiekt odtwarzacza na podstawie adresu danego zasobu.
- `public Player createPlayer(InputStream is, String` *`typ`*`)` — tworzy obiekt odtwarzacza na podstawie danego strumienia wejścia i określonego typu MIME.

Źródła plików multimedialnych

MMAPI 1.1 daje olbrzymie możliwości co do wyboru źródeł, z których można pobierać dane multimedialne. Najważniejszą rolę odgrywa URL, przekazywany w parametrze metody `createPlayer()`. Adres, jak każdy inny, składa się z trzech części. Jednak w przypadku lokalizatorów multimediów może on przyjąć postać daleko inną od tej znanej z codziennego użytkowania komputera.

Podstawowym typem jest odtwarzanie plików pobranych za pomocą protokołu HTTP. URL przyjmuje wtedy postać:

```
http://www.serwer.org/folder/plik.wav
```

gdzie `http://` to określenie protokołu, `www.serwer.org` — nazwa hosta (komputera, z którym program musi się połączyć), a `/folder/plik.wav` — ścieżka do pliku na serwerze. Ta postać jest znana; zupełnie inaczej wygląda jednak konstruowanie adresów w przypadku przechwytywania danych audio i wideo.

Aby utworzyć obiekt klasy `Player`, który umożliwi rejestrowanie jakichkolwiek danych, należy zastosować protokół `capture://`. Następnie należy podać rodzaj przechwytywanego materiału — `audio` lub `video`. Na tym nie koniec — po znaku zapytania można określić jego parametry techniczne, np. `rate` (częstotliwość próbkowania w przypadku dźwięku) lub `width` i `height` (rozmiary obrazu w przypadku wideo).

Oczywiście przechwytywanie materiału audio i wideo wymaga zastosowania dodatkowych kontrolek; są to odpowiednio: `RecordControl` i `VideoControl`. Omówię je w jednym z następnych podrozdziałów, w momencie gdy zostaną zastosowane w naszym projekcie.

Interfejs programu

Aby zrozumieć, dlaczego stosujemy taki, a nie inny sposób tworzenia interfejsu, trzeba najpierw omówić działanie programu. Po uruchomieniu programu użytkownik musi wybrać rodzaj źródła danych: internet, system plików lub RMS. Pozostałe dwie możliwości to przechwytywanie — audio lub wideo. W przypadku pobierania pliku z internetu sytuacja jest najprostsza — należy udostępnić użytkownikowi pole do wprowadzenia adresu. Zdecydowanie bardziej skomplikowane jest wykorzystanie systemu plików. Nasz MIDlet udostępnia bowiem minimenadżer plików, który umożliwia swobodne poruszanie się po strukturze katalogów. W przypadku zbioru rekordów program wyświetla listę wszystkich zarejestrowanych multimediów. Bez względu na sposób pobierania użytkownik dociera do formatki, która jest wyświetlana przy odtwarzaniu plików.

OdtwarzaczMIDlet.java

```
    private Odtwarzacz odtwarzacz;
    private MenadzerPlikow menadzer;
    private List menu;
    private List listaFile;
    private List listaRms;
    private Form formaHttp;
    private Form formaAparat;
    private Form formaOdtwarzacz;
    private List listaPrzechwytujaca;
    private Display ekran;
    private TextField poleUrl;
    private final String[] POLECENIA_PRZECHWYTYWANIA = new String[]
       {"Start","Stop","Odtworz","Zapisz"};
    private final String[] OPCJE = new String[]
    {"Odtworz plik z Internetu","Odtworz plik z urzadzenia","Odtworz plik z
    ↪RMS","Przechwyc audio", "Zrob zdjecie"};
   }
```

Lista zmiennych ujawnia częściowo zawartość projektu. Na początku są zadeklarowane dwa kluczowe obiekty: `odtwarzacz` i `menadzer`. Pierwszy z nich odpowiada za wszelkie kwestie związane z wykorzystaniem składników pakietu `javax.microedition.media`, a drugi — za obsługę systemu plików.

Nie powinna dziwić duża liczba list użytych w tym projekcie. Większość wyświetlaczy musi dać użytkownikowi możliwość wyboru — do tego najlepiej nadają się właśnie listy. Dwie z nich mają stałe elementy — są zadeklarowane w powyższym listingu. Pozostałe dwie wczytują swoją zawartość z systemu plików i RMS.

Większą część konstruktora MIDletu zajmują instrukcje tworzące obiekty wyświetlaczy i zaopatrujące je w polecenia. Pojawia się przy tym ciekawa konstrukcja:

OdtwarzaczMIDlet.java

```
package pl.helion.j2mepp.odtwarzacz;

import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;
public class OdtwarzaczMIDlet extends MIDlet implements CommandListener
{
 public OdtwarzaczMIDlet() throws Exception
 {
  menu = new List("Wybierz akcje:",Choice.IMPLICIT,OPCJE,null);
  Command wybierz = new Command("Wybierz",Command.OK,0);
  Command koniec = new Command("Koniec",Command.EXIT,0);
  Command powrot = new Command("Powrot",Command.EXIT,0);
  menu.addCommand(koniec);
  menu.addCommand(wybierz);
  menu.setSelectCommand(wybierz);
  menu.setCommandListener(this);
  formaHttp = new Form("Podaj adres URL:");
  poleUrl = new TextField("","http://",150,TextField.ANY);
  formaHttp.append(poleUrl);
  Command ok = new Command("OK",Command.OK,0);
  formaHttp.addCommand(ok);
  formaHttp.addCommand(powrot);
  formaHttp.setCommandListener(this);
  listaFile = new List("Wybierz plik:",List.IMPLICIT);
  Command wejdz = new Command("Wejdz",Command.ITEM,0);
  Command wyjdz = new Command("Wyjdz",Command.ITEM,1);
  listaFile.addCommand(wejdz);
  listaFile.addCommand(wyjdz);
  listaFile.addCommand(powrot);
  listaFile.setSelectCommand(wejdz);
  listaFile.setCommandListener(this);
  listaPrzechwytujaca = new List("Przechwyc
  ↪audio",Choice.IMPLICIT,POLECENIA_PRZECHWYTYWANIA,null);
  listaPrzechwytujaca.addCommand(powrot);
  listaPrzechwytujaca.addCommand(wybierz);
  listaPrzechwytujaca.setSelectCommand(wybierz);
  listaPrzechwytujaca.setCommandListener(this);
  listaRms = new List("Wybierz element:",Choice.IMPLICIT);
  listaRms.addCommand(wybierz);
  listaRms.addCommand(powrot);
  listaRms.setSelectCommand(wybierz);
  listaRms.setCommandListener(this);
  formaOdtwarzacz = new Form("Teraz odtwarzane...");
  formaOdtwarzacz.append("");
  formaOdtwarzacz.addCommand(powrot);
  formaOdtwarzacz.setCommandListener(this);
  formaAparat = new Form("Zrob zdjecie");
  formaAparat.append("");
  Command pstryk = new Command("Pstryk!",Command.OK,0);
  formaAparat.addCommand(powrot);
  formaAparat.addCommand(pstryk);
  formaAparat.setCommandListener(this);
  odtwarzacz = new Odtwarzacz(this);
  menadzer = new MenadzerPlikow(this);
```

```
 ekran = Display.getDisplay(this);
 ekran.setCurrent(menu);
}
```

W powyższym kodzie znajdują się dwie ciekawe, zastosowane po raz pierwszy konstrukcje. Polecenie o nazwie *Wybierz* pojawia się w aplikacji wiele razy. Nie ma sensu tworzyć takiego samego obiektu pod różnymi nazwami — jedna instancja klasy `Command` może być dodana do różnych formularzy, o ile przy identyfikowaniu polecenia korzysta się z jego typu i priorytetu. Mimo że metoda `setSelectCommand()` jednocześnie dodaje polecenie (o ile nie zostało wcześniej jawnie dodane), to obiekt `wybierz` jest dodawany ręcznie w kodzie MIDletu, aby lepiej zobrazować liczbę zastosowanych do każdej formatki poleceń.

Drugim intrygującym mechanizmem jest dodanie do formatek `formaOdtwarzacz` i `forma` ➥`Aparat` pustych etykiet tekstowych. Tak naprawdę rodzaj dodanego komponentu nie ma znaczenia — wywołanie tego wariantu metody `append()` jest po prostu najkrótsze. Ważne jest, aby formatka miała jeden komponent. W trakcie działania aplikacji owa pusta etykieta tekstowa zostanie zastąpiona np. kontrolką wyświetlającą film.

Przed omówieniem najdłuższej metody zajmę się krótkimi metodami pomocniczymi, wykorzystywanymi przez pozostałe klasy pakietu do działań na interfejsie MIDletu:

OdtwarzaczMIDlet.java

```
 public void startApp() {}
public void pauseApp() {}
public void destroyApp(boolean u)
{
 odtwarzacz.koniec();
}
public void wyswietlElementy(String[] wartosci)
{
 listaFile.deleteAll();
 for (int i=wartosci.length-1;i>=0;i--)
  listaFile.append(wartosci[i],null);
}
public void wlaczWyswietlacz(Item it)
{
 if (it!=null)
  formaOdtwarzacz.set(0,it);
 else
  formaOdtwarzacz.set(0,new StringItem("",""));
 ekran.setCurrent(formaOdtwarzacz);
}
```

W momencie zakończenia aplikacji odtwarzacz musi zwolnić wszystkie swoje zasoby. Dwie pozostałe metody przypominają te znane z poprzednich projektów. W metodzie `wyswietlElementy()` dodajemy nowe elementy od końca tablicy, tak aby ostatni element z tablicy znalazł się na górze wyświetlanej listy. Druga z metod pełni kluczową rolę przy odtwarzaniu filmów. Obiekt `it` zawiera obszar, w którym jest wyświetlany film. Musi on zatem zostać wyświetlony na formatce. Przy odtwarzaniu dźwięków film nie jest jednak potrzebny, dlatego stosuję pustą etykietę tekstową.

Metoda obsługi zdarzeń w tym projekcie jest rozbudowana. Wynika to z dużej liczby zawartych w aplikacji wyświetlaczy i dostępnych poleceń. Jej opis jest podzielony na kilka części:

OdtwarzaczMIDlet.java

```
public void commandAction(Command c, Displayable s)
 {
  if (s == menu)
  {
   if (c.getCommandType() == Command.OK)
   {
    if (menu.getSelectedIndex() == 0)
     ekran.setCurrent(formaHttp);
    if (menu.getSelectedIndex() == 1)
    {
     menadzer.odswiez();
     menadzer.wyswietlKorzenie();
     ekran.setCurrent(listaFile);
    }
    if (menu.getSelectedIndex() == 2)
    {
     listaRms.deleteAll();
     String[] numery = odtwarzacz.pobierzID();
     for (int i=0;i<numery.length;i++)
      listaRms.append(numery[i],null);
     ekran.setCurrent(listaRms);
    }
    if (menu.getSelectedIndex() == 3)
     ekran.setCurrent(listaPrzechwytujaca);
    if (menu.getSelectedIndex() == 4)
    {
     ekran.setCurrent(formaAparat);
     Item it = odtwarzacz.pobierajObraz();
     if (it!=null)
      formaAparat.set(0,it);
    }
   }
   if (c.getCommandType() == Command.EXIT)
   {
    this.destroyApp(true);
    this.notifyDestroyed();
   }
  }
```

W metodzie tej można znaleźć wiele odwołań do obiektów `menadzer` i `odtwarzacz`. Istotą obsługi zdarzeń listy-menu jest wyświetlanie innych formatek. Niektóre z nich wymagają jednak wcześniejszego przygotowania. Tak jest w przypadku listy `lista ➥Plikow`, która wyświetla listę katalogów i plików. Przed pierwszym wyświetleniem program musi pobrać listę korzeni systemu plików (ang. *root*) — szerzej zostanie to omówione nieco dalej. Nie inaczej jest, gdy pobieramy spis nagrań z RMS. Po uprzednim wyczyszczeniu listy i pobraniu identyfikatorów następuje wyświetlenie elementów. Wreszcie formatka `formaAparat` otrzymuje obiekt wyświetlający obraz z kamery telefonu i ustawia go jako komponent — teraz widoczne jest zastosowanie jednej z pustych etykiet tekstowych. Obsługa zdarzeń kolejnych wyświetlaczy jest zróżnicowana:

OdtwarzaczMIDlet.java

```
if (s == formaHttp)
  {
   if (c.getCommandType() == Command.OK)
    if (!poleUrl.getString().equals(""))
    {
     odtwarzacz.przygotuj(poleUrl.getString());
    }
   if (c.getCommandType() == Command.EXIT)
    ekran.setCurrent(menu);
  }
  if (s == listaFile)
  {
   if (c.getCommandType() == Command.ITEM)
   {
    try
    {
     if (c.getPriority()==0)
     {
      int k = listaFile.getSelectedIndex();
      if (k>-1)
      {
       menadzer.przejdzDo(listaFile.getString(k));
       if (menadzer.jestKatalog())
       {
        String[] wyniki = menadzer.zwrocZawartosc();
        this.wyswietlElementy(wyniki);
       } else
       {
        odtwarzacz.przygotuj(menadzer.pobierzSciezke());
       }
      }
     }
     if (c.getPriority()==1)
     {
      menadzer.wyjdzDoGory();
      String[] wyniki = menadzer.zwrocZawartosc();
      this.wyswietlElementy(wyniki);
     }
    }
    catch (Exception e)
    {
     e.printStackTrace();
    }
   }
   if (c.getCommandType() == Command.EXIT)
   {
    ekran.setCurrent(menu);
   }
  }
```

Obsługa formatki wczytującej plik z internetu jest banalna — wszystkim zajmuje się metoda `przygotuj()`. Zdecydowanie bardziej złożona jest konstrukcja obsługująca system plików. Polecenia są tylko dwa: *Wejdz* (priorytet 0) i *Wyjdz* (priorytet 1). Pierwsze z nich jest gotowe zarówno na sytuację, w której zaznaczony obiekt jest folderem, jak i plikiem. Metoda `zwrocZawartosc()` pobiera listę katalogów i plików aktualnego

katalogu, określonego za pomocą metody `przejdzDo()`. Jeśli mamy do czynienia z plikiem, wtedy próbujemy go odtworzyć. Rozszerzenie tego modułu, aby sprawdzał zawartość plików przed próbą odtworzenia, czytelnik może potraktować jako ćwiczenie rozszerzające funkcjonalność programu. Prostszą czynnością jest przejście do katalogu wyższego rzędu.

OdtwarzaczMIDlet.java

```
if (s == listaRms)
  {
   if (c.getCommandType() == Command.OK)
    if (listaRms.getSelectedIndex()>-1)
     odtwarzacz.przygotuj(" rms://"+listaRms.getString
     ➥(listaRms.getSelectedIndex()));
   if (c.getCommandType() == Command.EXIT)
    ekran.setCurrent(menu);
  }
  if (s == formaOdtwarzacz)
  {
   if (c.getCommandType() == Command.EXIT)
   {
    ekran.setCurrent(menu);
    odtwarzacz.przerwij();
   }
  }
  if (s == formaAparat)
  {
   if (c.getCommandType() == Command.OK)
   {
    try
    {
     odtwarzacz.pobierzZdjecie();
    }
    catch (Exception e)
    {
     e.printStackTrace();
    }
   }
   if (c.getCommandType() == Command.EXIT)
   {
    ekran.setCurrent(menu);
    odtwarzacz.przerwij();
   }
  }
```

Następna lista, przechowująca spis nagranych multimediów, korzysta w adresie z protokołu o nazwie `"rmsp"`, utworzonego na potrzeby tej aplikacji. W rzeczywistości chodzi o sprecyzowanie jednego systemu przekazywania danych do metody `przygotuj()`. Obiekt menedżera pobiera identyfikator rekordu, który znajduje się za definicją protokołu, a następnie na jego podstawie wczytuje właściwy rekord. Forma `formaOdtwarzacz` musi zadbać o przerwanie odtwarzania w chwili, gdy użytkownik wybierze polecenie *Wyjdz*. Utworzenie zdjęcia za pomocą formatki `formaAparat` wymaga wywołania tylko jednej metody — wszystkie szczegóły implementacyjne są zawarte w klasach `Odtwarzacz` i `CzytnikDanych`.

OdtwarzaczMIDlet.java

```
if (s == listaPrzechwytujaca)
  {
   if (c.getCommandType() == Command.OK)
   {
    if (listaPrzechwytujaca.getSelectedIndex() == 0)
     odtwarzacz.przechwyc(true);
    if (listaPrzechwytujaca.getSelectedIndex() == 1)
     odtwarzacz.przechwyc(false);
    if (listaPrzechwytujaca.getSelectedIndex() == 2)
     odtwarzacz.odtworzNagranie();
    if (listaPrzechwytujaca.getSelectedIndex() == 3)
     try
     {
      odtwarzacz.zapisz("audio/x-wav");
     }
     catch (Exception e)
     {
      e.printStackTrace();
     }
   }
   if (c.getCommandType() == Command.EXIT)
   {
    odtwarzacz.przechwyc(false);
    ekran.setCurrent(menu);
   }
  }
 }
```

Ostatnia z list udostępnia sporą paletę czynności. Metoda `przechwyc()` rozpoczyna lub zatrzymuje nagrywanie, zgodnie z wartością przekazanego parametru. Warto zwrócić uwagę na metodę `zapisz()` z klasy `Odtwarzacz`. Operacja zapisu wymaga podania typu MIME, jaki ma być określony dla nagrania. Na jego podstawie można wywnioskować, w jaki sposób multimedia stworzone przez użytkownika są przechowywane w RMS. Otóż każdy rekord składa się z dwóch części: zawartości oraz nazwy typu MIME. Dzięki temu można przechowywać różnego rodzaju zawartość, która dzięki zapamiętanemu typowi MIME może być prawidłowo rozpoznana przez odtwarzacz.

W trakcie testowania nie należy martwić się obrazem, który znajduje się na ekranie emulatora — w prawdziwym urządzeniu w tym miejscu znajdowałby się aktualny obraz kamery.

Odtwarzacz a FileConnection Optional Package

Zanim omówię multimedialny aspekt naszej aplikacji, postaram się przybliżyć funkcjonowanie systemu plików w telefonach komórkowych. Następnie opiszę klasę `Menadzer ↪Plikow`, która w naszym MIDlecie odpowiada za przeglądanie struktury katalogów i wybór plików do odtwarzania.

Prawie jak PC

API definiujące wykonywanie operacji na plikach jest zawarte w JSR-75. Pakiet odpowiedzialny za obsługę plików to `javax.microedition.io.file`. Zawiera on pięć elementów, jednak z punktu widzenia naszej aplikacji najważniejsze są dwa:

- interfejs `FileConnection` umożliwiający otwieranie plików, przeglądanie zawartości katalogów i inne podstawowe operacje,
- klasa `FileSystemRegistry`, która m.in. udostępnia listę korzeni systemu plików urządzenia.

Dwa razy w tym rozdziale pojawiło się słowo „korzeń". Jak sama nazwa wskazuje, jest on czymś podstawowym; można porównać go do partycji systemu Windows (np. *c:*, *d:*). Korzenie mogą wskazywać na foldery znajdujące się w pamięci telefonu, ale mogą też dotyczyć np. udostępnionych kart pamięci[1].

Obecność powyższych elementów JSR-75 w pakiecie `javax.microedition.io.file` oraz charakterystyczna nazwa interfejsu pozwalają przypuszczać, że proces korzystania z systemu plików jest podobny do nawiązywania połączenia z internetem. Tak faktycznie jest; jedyną różnicą, poza stosowanym interfejsem, jest składnia adresu, za pomocą którego są lokalizowane pliki i katalogi. Przykładowy adres wygląda tak:

```
file:///root1/filmy/film.mpg
```

Pierwsze sześć znaków, czyli `file://`, stanowi określenie protokołu. Następnie widzimy nazwę korzenia (`/root1`) oraz ścieżkę do pliku wraz z jego nazwą (`/filmy/film.mpg`).

Nawiązać połączenie z danym plikiem lub katalogiem można nawet wtedy, gdy on nie istnieje. Można go wtedy utworzyć za pomocą metody `create()` lub `mkdir()`. Jeśli jednak wczytywany zasób istnieje, sprawa jest prosta. W przypadku pliku wystarczy odwołać się do wiedzy z rozdziału 4. i wywołać metodę `openInputStream()`. Trochę bardziej skomplikowanie wygląda sytuacja, gdy mamy do czynienia z katalogami. Można wywołać wtedy metodę `list()`, która zwraca listę wszystkich katalogów i plików w określonej lokalizacji.

Jednak co powinniśmy zrobić, gdy chcemy wczytać nowy plik lub sprawdzić zawartość innego katalogu? Chociaż w interfejsie `FileConnection` jest zadeklarowana metoda `setFileConnection()`, to jest ona obwarowana licznymi zastrzeżeniami (m.in. element aktualnie wskazywany musi być katalogiem, a nowy element musi istnieć w tym katalogu). Dlatego zaleca się tworzenie nowego obiektu interfejsu `FileConnection` przy dostępie do każdego kolejnego elementu.

FileConnection Optional Package a uprawnienia dostępu

Nie każde urządzenie zezwala na pełny zakres operacji w odniesieniu do udostępnianych plików i katalogów. Tradycyjnie zakres żądanych uprawnień określa się w metodzie `open()` klasy `Connector`. Stałe definiujące sposób dostępu są identyczne jak w przypadku

[1] Na przykład w telefonach wykorzystujących system *Nokia OS* pamięć wewnętrzna jest widziana jako *C:*, a dodatkowa karta pamięci jako *E:*.

połączeń internetowych. W przypadku niemożności uzyskania żądanego trybu dostępu do pliku lub katalogu aplikacja zwraca wyjątek klasy `SecurityException`.

Tryb dostępu ma wpływ na możliwość wykonania metod klasy `FileConnection`. Dlatego należy uważać, czy w trybie tylko do odczytu (`READ_ONLY`) nasza aplikacja nie wywołuje metody `openOutputStream()` — taka operacja, jakkolwiek zostanie dopuszczona przez kompilator, na pewno spowoduje wyjątek `SecurityException`.


Wskazówka

Nie należy uruchamiać dwóch instancji tego samego emulatora wykorzystujących FCOP API, ponieważ może dojść do błędów w wykonywaniu operacji odczytu i zapisu.

Opis możliwości FCOP

Metody udostępniane przez klasę i interfejs znajdują się w poniższym zestawieniu.

`javax.microedition.io.file.FileSystemRegistry`

- `public static Enumeration listRoots()` — zwraca wszystkie dostępne w urządzeniu korzenie. Nazwy są przechowywane w postaci łańcuchów w obiekcie wyliczeniowym.

`javax.microedition.io.file.FileConnection`

- `public long availableSize()` — zwraca ilość dostępnego miejsca w korzeniu, w którym znajduje się plik lub katalog określony danym obiektem połączenia.
- `public void create()` — tworzy plik o ścieżce określonej w danym obiekcie połączenia.
- `public void delete()` — usuwa plik o ścieżce określonej w danym obiekcie połączenia.
- `public boolean exists()` — sprawdza, czy plik lub katalog określony w danym obiekcie połączenia istnieje.
- `public boolean isDirectory()` — sprawdza, czy obiekt określony w połączeniu jest katalogiem.
- `public Enumeration list(String` *`klucz`*`, boolean` *`czyUkryte`*`)` — zwraca wszystkie pliki i katalogi znajdujące się w katalogu określonym w danym obiekcie połączenia. Wszystkie elementy są filtrowane według klucza (można zastosować znak *, oznaczający dowolny ciąg znaków); jeśli parametr *`czyUkryte`* ma wartość `true`, metoda zwraca także pliki i katalogi ukryte.


Wskazówka

Jeśli emulator nie zwraca poprawnej zawartości katalogu, należy skasować plik *in.use* z katalogu *<KATALOG>/appdb/<emulator>*.

- `public void mkdir()` — tworzy katalog o ścieżce określonej w danym obiekcie połączenia.

- `public InputStream openInputStream()` — zwraca strumień wejścia (do odczytu) dla określonego pliku.
- `public OutputStream openOutputStream()` — zwraca strumień wyjścia (do zapisu) dla określonego pliku.
- `public void setFileConnection(String nazwa)` — tworzy w danym obiekcie połączenie z nowym plikiem lub katalogiem i zastępuje nim aktualne.
- `public void truncate(long n)` — usuwa wszystkie dane z pliku określonego w połączeniu, począwszy od *n*-tego bajtu.

Implementacja przeglądarki systemu plików w projekcie

Podsumujmy wnioski, które można wyciągnąć z analizy MIDletu, a zwłaszcza metody `commandAction()`, dotyczące wykorzystania w odtwarzaczu systemu plików:

- Program ma możliwość pobrania listy korzeni.
- Program może poruszać się po strukturze katalogów w obydwie strony (w głąb i do góry).
- Program może pobrać plik i udostępnić go obiektowi odtwarzacza.

Wszystkie metody potrzebne do wykonania powyższych czynności są zawarte w klasie `MenadzerPlikow`.

Lista zmiennych klasy jest wyjątkowo krótka:

MenadzerPlikow.java

```
    private FileConnection plik;
    private OdtwarzaczMIDlet m;
    private String sciezka = "/";
    private static final String PRZED = "file://";
   }
```

Znaczenia zmiennej `plik` dla działania klasy nie trzeba chyba tłumaczyć. Zmienna `sciezka` w połączeniu ze stałą `PRZEDROSTEK` tworzy pełną ścieżkę dostępu do pliku i katalogu. Obiekt MIDletu jest potrzebny do wywołania metod odświeżających listę, która przedstawia zawartość aktualnego katalogu (`listaPlikow`).

Konstruktor zawiera bardziej interesującą konstrukcję. Stosowane jest w nim sprawdzenie, czy urządzenie oferuje dostęp do File API:

MenadzerPlikow.java

```
package pl.helion.j2mepp.odtwarzacz;

import javax.microedition.io.file.*;
import javax.microedition.io.*;
import java.util.*;
```

```
import java.io.*;
public class MenadzerPlikow
{
 public MenadzerPlikow(OdtwarzaczMIDlet _m) throws Exception
 {
  m = _m;
  String v = System.getProperty("microedition.io.file.FileConnection.version");
  if (v==null)
   throw new Exception("Brak obslugi systemu plikow!");
 }
```

Metoda `getProperty()` klasy `System` służy do pobierania właściwości maszyny wirtualnej i zwraca `null`, jeśli właściwość o podanej nazwie nie istnieje. Jeśli tak się stanie w naszym przypadku, zostanie wygenerowany wyjątek. Pierwsza z metod to metoda `wyswietlKorzenie()`:

MenadzerPlikow.java

```
 public void wyswietlKorzenie()
 {
  new Thread(new Runnable(){
   public void run()
   {
    Enumeration zestaw = FileSystemRegistry.listRoots();
    String[] rooty = przerobEnumerationNaString(zestaw);
    m.wyswietlElementy(rooty);
   }
  }).start();
 }
```

W powyższej metodzie po raz pierwszy została zastosowana konstrukcja tworząca i uruchamiająca nowy wątek w taki sposób. Korzystamy z jednego z konstruktorów klasy `Thread`:

```
public Thread(Runnable watek)
```

oraz z faktu, że Java umożliwia utworzenie anonimowego obiektu interfejsu, o ile zostaną zadeklarowane wszystkie jego metody. W naszej sytuacji wystarczy utworzyć metodę `run()`. Dlaczego jednak tworzymy dla tej czynności nowy wątek?

Wykonanie niektórych czynności, zwłaszcza związanych z wykorzystaniem zewnętrznych zasobów, wymaga zezwolenia. Uzyskuje się je na dwa sposoby:

- przez cyfrowe podpisanie MIDletu; wymaga to jednak uzyskania certyfikatu autentyczności z któregoś z dozwolonych centrów autentykacji — jest to proces stosunkowo długi i drogi, zwłaszcza w Polsce;
- przez wyświetlenie komunikatu i bezpośrednią zgodę użytkownika.

Jeśli użytkownik zezwoli na daną czynność, aplikacja wykonuje kolejne instrukcje; w przeciwnym razie zgłaszany jest wyjątek klasy `SecurityException`. Niestety, emulator oraz niektóre modele telefonów na pytanie o dostęp reagują zawieszeniem programu, gdy pytanie pojawia się w metodzie obsługi poleceń. W związku z tym wszystkie metody, które mogą spowodować wyświetlenie komunikatu, powinny być wywoływane w nowych wątkach.

Prośba o pozwolenie jest wyświetlana zazwyczaj tylko za pierwszym razem — później program pamięta decyzję użytkownika. W związku z tym nie trzeba zabezpieczać wszystkich metod. W klasie `MenadzerPlikow` wiadomo, że to metoda `wyswietlKorzenie()` zawsze jako pierwsza prosi o dostęp, tak więc tylko ona musi być uruchamiana w nowym wątku.

Metoda ta wykorzystuje metodę `listRoots()` klasy `FileSystemRegistry`. Przy użyciu pomocniczej metody `przerobEnumerationNaString()` program uzyskuje tablicę łańcuchów z obiektu wyliczeniowego. Dysponując tablicą nazw korzeni, można wyświetlić je za pomocą metody `wyswietlElementy()`.

Metoda `przerobEnumerationNaString()` wykorzystuje wektor:

MenadzerPlikow.java

```
public String[] przerobEnumerationNaString(Enumeration e)
{
 Vector lista = new Vector();
 while (e.hasMoreElements())
  lista.addElement(e.nextElement());
 String[] wyniki = new String[lista.size()];
 for (int i=0;i<wyniki.length;i++)
  wyniki[i] = (String)lista.elementAt(i);
 return wyniki;
}
```

Na początku przekształcamy obiekt wyliczeniowy na wektor, aby następnie zamienić go na tablicę łańcuchów. Dlaczego wykorzystujemy dodatkowy wektor do utworzenia tablicy? Niestety, klasa `Enumeration` nie ma metody zwracającej liczbę obiektów znajdujących się w danym obiekcie wyliczeniowym. Nie znając tej liczby, nie można utworzyć tablicy. Z kolei klasa `Vector` taką metodę ma (`size()`).

Następne metody wykorzystywane w klasie MIDletu odpowiadają za poruszanie się po strukturze katalogów. Są to metody `przejdzDo()` i `wyjdzDoGory()`:

MenadzerPlikow.java

```
public void przejdzDo(String nazwa) throws Exception
 {
  sciezka += nazwa;
  this.ustalPlik();
 }
 public void wyjdzDoGory() throws Exception
 {
  if (sciezka.length()>1)
  {
   if (this.jestKatalog())
   {
    sciezka = sciezka.substring(0,sciezka.length()-1);
    int indeks = sciezka.lastIndexOf('/');
    sciezka = sciezka.substring(0,indeks+1);
   } else
   {
    int indeks = sciezka.lastIndexOf('/');
    sciezka = sciezka.substring(0,indeks+1);
   }
```

```
  if (sciezka.length()>1)
   this.ustalPlik();
 }
}
```

Metoda wchodząca w głąb struktury katalogów jest uniwersalna. Dzięki temu składa się tylko z dwóch linijek kodu. Po zmianie bieżącej ścieżki wywoływana jest metoda `ustalPlik()`. To właśnie ona tworzy nowy obiekt połączenia z plikiem lub katalogiem. Druga z metod zawiera jedynie operacje na łańcuchach. Ścieżki do katalogu i pliku różnią się jednym, ale ważnym detalem: pierwsza z nich zawiera na końcu znak / (ukośnik). W przypadku katalogu w pierwszej instrukcji pozbywamy się właśnie tego końcowego ukośnika. Następnie znajdujemy ostatni ukośnik i usuwamy wszystko, co się za nim znajduje (czyli nazwę katalogu, z którego chcemy wyjść). W przypadku pliku sposób postępowania jest podobny, z wyjątkiem usuwania końcowego ukośnika.

Kluczową metodą tej klasy jest metoda `ustalPlik()`. Nawiązuje ona połączenie z plikiem i sprawdza, czy plik istnieje. Jeśli nie — zwraca wyjątek:

MenadzerPlikow.java

```
private void ustalPlik() throws IOException
{
 if (plik!=null)
  plik.close();
 plik = (FileConnection)Connector.open(this.pobierzSciezke(),Connector.READ);
 if (!plik.exists())
  throw new IOException("Brak pliku!");
}
```

Parametr przekazany metodzie `open()` jest pobierany za pomocą pomocniczej metody, wprowadzonej, aby budowanie pełnego adresu odbywało się w jednym miejscu w kodzie.

Metoda `zwrocZawartosc()` to ostatnia ważna i funkcjonalna metoda w tej klasie:

MenadzerPlikow.java

```
public String[] zwrocZawartosc() throws IOException
{
 if (sciezka.length()==1)
  return this.przerobEnumerationNaString(FileSystemRegistry.listRoots());
 this.ustalPlik();
 if (plik.isDirectory())
 {
  Enumeration list = plik.list();
  String[] wyniki = this.przerobEnumerationNaString(list);
  return wyniki;
 } else
  return null;
}
```

Jeśli program chce zwrócić listę elementów początkowego katalogu (który zawiera zbiór korzeni), trzeba wywołać metodę `listRoots()`. W przeciwnym wypadku metoda ustala obecną ścieżkę. Następnie, w przypadku gdy bieżąca ścieżka prowadzi do katalogu, zwracana jest lista jego elementów. W przeciwnym razie zwracana jest wartość `null`.

Ostatnie trzy metody mają charakter pomocniczy. Ich treść można wydedukować na podstawie wcześniejszych metod i FCOP API:

MenadzerPlikow.java

```
public String pobierzSciezke()
{
 return PRZED+sciezka;
}
public void odswiez()
{
 sciezka = "/";
}
public boolean jestKatalog()
{
 if (plik!=null)
  return plik.isDirectory();
 else
  return false;
}
```

Metoda `pobierzSciezke()` łączy dwie kluczowe części adresu: protokół i właściwą ścieżkę. Metoda `odswiez()` ma za zadanie przywrócić domyślną wartość ścieżki (przy powtórnym przeglądaniu zawartości systemu plików). Ostatnia metoda sprawdza, czy plik istnieje, i dopiero wtedy zwraca wartość metody `isDirectory()`. W przeciwnym wypadku zwraca wartość `false`.

Obsługa multimediów w odtwarzaczu

Nieuchronnie zbliżamy się do momentu, gdy omówię największą klasę, z jaką dotąd się spotkaliśmy. Klasa `Odtwarzacz`, bo o niej mowa, zawiera wszystkie funkcje związane z pakietem `javax.microedition.media`. Proces obsługi danych został wydzielony do klasy `CzytnikDanych`.

Na wstępie bardzo ważna informacja: nasz odtwarzacz nie implementuje wielu kontrolek, które są związane z procesem odtwarzanialnych. Moim celem w tym projekcie było zaprezentowanie istoty odtwarzania treści multimedialnych — obsługa poszczególnych kontrolek jest prosta i sprowadza się w dużej mierze do zapoznania się z ich metodami.

Zacznę, jak zwykle, od listy zmiennych — również dość obszernej i zróżnicowanej:

Odtwarzacz.java

```
private Player p;
private OdtwarzaczMIDlet m;
private VideoControl vc;
private VideoControl aparat;
private RecordControl rc;
private ByteArrayOutputStream baos;
private String sciezka;
private CzytnikDanych czytnik;
private byte[] bufor = new byte[0];
```

```
 private boolean tryb;
 private boolean nagrywa = false;
 private String typZdjecia;
}
```

Obiekt `p` jest najczęściej wykorzystywany w całej klasie — to on pełni rolę odtwarzacza. Zastanawiać mogą aż dwie kontrolki wideo: `vc` i `aparat`. Jest to spowodowane koniecznością użycia osobnych obiektów do odtwarzania i przechwytywania obrazu z aparatu. Znaczenie pozostałych zmiennych będę przybliżać przy omawianiu konkretnych metod.

Należy zwrócić uwagę na obiekt klasy `CzytnikDanych`, gdyż to właśnie w nim będzie zawarty proces wczytywania danych. Po raz pierwszy stosujemy go już w konstruktorze:

Odtwarzacz.java

```
package pl.helion.j2mepp.odtwarzacz;

import javax.microedition.media.*;
import javax.microedition.media.control.*;
import java.io.*;
import javax.microedition.lcdui.*;

public class Odtwarzacz
{
 public Odtwarzacz(OdtwarzaczMIDlet p_m)
 {
  m = p_m;
  czytnik = new CzytnikDanych(p_m);
  typZdjecia = this.pobierzDomyslnyTyp();
 }
```

Obiekt czytnika również wymaga odwołania do klasy MIDletu — ze względu na konieczność zamknięcia aplikacji w przypadku błędu pobierania danych — ale to omówię szczegółowo w swoim czasie. Zmienna `typZdjecia` przechowuje identyfikator formatu, w jakim zapisywane będą zdjęcia z aparatu. Pomocnicza metoda `pobierzDo`
`↪myslnyTyp()` wykorzystuje właściwość maszyny wirtualnej o nazwie `video.snapshot.`
`↪encodings`:

Odtwarzacz.java

```
public String pobierzDomyslnyTyp()
 {
  String typy = System.getProperty("video.snapshot.encodings");
  if (typy.indexOf("jpeg")>-1)
   return "encoding=jpeg";
  if (typy.indexOf("png")>-1)
   return "encoding=png";
  if (typy.indexOf("gif")>-1)
   return "encoding=gif";
  return null;
 }
```

Ponownie wykorzystujemy metodę `System.getProperty()`. Łańcuch `typy` przybiera następującą postać:

```
encoding=jpeg encoding=png
```

Analogicznie, parametr określający typ zdjęcia dla wykonującej je metody musi mieć format `encoding=xxx`, gdzie `xxx` to nazwa typu. Jak widać, właściwość `video.snap➥shot.encodings` zawiera szereg poprawnie określonych kodowań; trzeba tylko wybrać typ. Priorytety wybieranych formatów zależą od kolejności instrukcji `if` w metodzie. Mimo że emulator obsługuje dwa typy zdjęć, instrukcja sprawdzająca typ `jpeg` występuje jako pierwsza i to właśnie przy użyciu typu `JPEG` zdjęcie zostanie zapisane.

Najczęściej pojawiającą się w kodzie MIDletu metodą jest `przygotuj()`. Pobiera ona za pomocą obiektu `czytnik` dane wymagane przez odtwarzacz, a następnie rozpoczyna proces odtwarzania:

Odtwarzacz.java

```
public void przygotuj(String p_sciezka)
 {
  sciezka = p_sciezka;
  new Thread(new Runnable(){
   public void run()
   {
    try
    {
     p = czytnik.analizuj(sciezka);
     if (p!=null)
     {
      p.realize();
      if (p.getContentType().equals("video/mpeg"))
       tryb = true;
      else
       tryb = false;
      odtwarzaj();
     } else
     if (jestObrazek(czytnik.getTyp()))
     {
      Image obraz = Image.createImage(czytnik.getStrumien());
      ImageItem it = new ImageItem("",obraz,ImageItem.LAYOUT_CENTER,"");
      m.wlaczWyswietlacz(it);
     }
    }
    catch (Exception e)
    {
     e.printStackTrace();
    }
   }
  }).start();
 }
```

Na początku musimy przygotować obiekt odtwarzacza, czym zajmuje się specjalna metoda klasy `Odtwarzacz`. Jeśli obiekt odtwarzacza nie obsługuje danego medium, zwracana jest wartość `null` i możemy sprawdzić, czy dane źródło nie określa obrazka. Jeśli tak — pobieramy jego strumień i możemy wyświetlić go za pomocą komponentu `ImageItem`. Jeśli zaś odtwarzacz napotka na „klasyczny” plik multimedialny, należy określić rodzaj medium — audio lub wideo — i przystąpić do odtwarzania.

Wywołania metod klasy `Odtwarzacz` w anonimowym obiekcie interfejsu `Runnable` nie zawierają słowa kluczowego `this` (np. `odtwarzaj()` zamiast `this.odtwarzaj()`), ponieważ słowo `this` użyte w tym kontekście oznaczałoby odwołanie do obiektu interfejsu `Runnable`, a nie do obiektu klasy zewnętrznej.

Zajmijmy się teraz metodą `odtwarzaj()`, która jest wywoływana w metodzie `przygotuj()`:

Odtwarzacz.java

```
public boolean odtwarzaj()
 {
  Item it = null;
  if (tryb && p!=null && p.getState() == Player.REALIZED)
  {
   vc = (VideoControl)p.getControl("VideoControl");
   if (vc!=null)
   {
    it = (Item)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE,null);
   }
  }
  if (p!=null && p.getState() == Player.REALIZED)
  {
   try
   {
    m.wlaczWyswietlacz(it);
    p.prefetch();
    p.start();
    return true;
   }
   catch (MediaException me)
   {
    me.printStackTrace();
   }
   return false;
  } else
   return false;
 }
```

Powyższa metoda swoje działanie uzależnia od zmiennej `tryb`. Gdy mamy do czynienia z plikiem wideo (`tryb=true`), metoda przypisuje do komponentu `it` wyświetlacz wideo. Odtwarzacz będący w stanie `REALIZED` może zwrócić kontrolkę interfejsu `VideoControl`. Następnie wywoływana jest metoda `initDisplayMode()`, która zwraca obiekt wyświetlacza wideo. Po wykonaniu opcjonalnej części metody związanej z odtwarzaniem wideo metoda próbuje ustawić komponent graficzny (jeśli odtwarzany jest plik audio, wtedy `it==null` i metoda `wlaczWyswietlacz()` nie wykona żadnej operacji), a następnie rozpoczyna odtwarzanie pliku.

Odtwarzanie, zwłaszcza plików wideo, to proces wymagający użycia dużej ilości pamięci. Należy pamiętać, że odtwarzanie dużych plików wideo może spowodować błąd krytyczny maszyny wirtualnej i w konsekwencji przerwanie działania aplikacji.

`javax.microedition.media.control.VideoControl`

- `public Object initDisplayMode(int tryb, Object argument)` — zwraca obiekt zawierający wyświetlany obraz. Wykorzystuje określony parametrem tryb (`USE_GUI_PRIMITIVE` albo `USE_DIRECT_VIDEO`) wraz z dodatkowym argumentem, którego znaczenie zależy od wartości pierwszego parametru:
  - `USE_GUI_PRIMITIVE` — w tym trybie metoda zwróci obiekt, który może stanowić element GUI; w praktyce oznacza to, że będzie on dziedziczył z klasy `Item` i będzie go można dodać do formy.
  - `USE_DIRECT_VIDEO` — w tym trybie `argument` musi być obiektem klasy `Canvas` (lub dziedziczącym z niej), a metoda zwraca `null`; wyświetlany obraz jest bezpośrednio rysowany w obszarze podanego obiektu.

`javax.microedition.media.Manager`

- `public static String[] getSupportedContentTypes(String protokol)` — zwraca listę typów MIME obsługiwanych przez odtwarzacz dla danego protokołu.

Nagrywanie dźwięku

W kodzie klasy MIDletu dwa razy jest wykorzystywana metoda `przechwyc()`. W przypadku rejestrowania dźwięku lub obrazu pozwolenie na wykonanie takiej czynności jest jednorazowe. Każda kolejna próba nagrania powoduje wyświetlenie pytania o pozwolenie (tak dzieje się w przypadku emulatora; poszczególne modele telefonów mogą różnić się pod tym względem). W związku z tym próba rozpoczęcia nagrania wymaga umieszczenia w nowym wątku:

Odtwarzacz.java

```
public void przechwyc(boolean czyStart)
{
 try
 {
  if (czyStart)
  {
   if (!nagrywa)
     new Thread(new Runnable(){
  public void run()
  {
   try
   {
     baos = new ByteArrayOutputStream();
     p = Manager.createPlayer("capture://audio");
     p.realize();
     rc = (RecordControl)p.getControl("RecordControl");
     rc.setRecordStream(baos);
     rc.startRecord();
     p.start();
     nagrywa = true;
    } catch (Exception e){}
```

```
 }
}).start();
      } else
      {
  if (nagrywa)
  {
     nagrywa = false;
           rc.commit();
     p.close();
     bufor = baos.toByteArray();
  }
 }
}
catch (Exception e)
{
  e.printStackTrace();
}
     }
```

Metoda `przechwyc()` obsługuje zarówno rozpoczęcie, jak i zakończenie nagrania. Dodatkową rolę odgrywa zmienna klasy `nagrywa`; określa ona stan, w jakim znajduje się odtwarzacz. Dzięki temu część zatrzymująca nie zostanie wywołana w stanie zatrzymania (`PREFETCHED`) i odwrotnie.

Jak widać, proces nagrywania jest realizowany z użyciem zarówno odtwarzacza, jak i kontrolki nagrywającej, `RecordControl`. Po utworzeniu kontrolki w klasyczny sposób — przy użyciu metody `getControl()` odtwarzacza znajdującego się w stanie `REALI`
`↪ZED` — należy wywołać metodę `setRecordStream()`, która określa strumień odbierający dane z kontrolki nagrywającej. W tym momencie nie pozostaje nic innego, jak rozpocząć nagrywanie i uruchomić odtwarzacz.

Proces kończenia nagrywania jest prostszy: należy zakończyć działanie kontrolki nagrywającej i odtwarzacza, a następnie wykorzystać dane ze strumienia (my kopiujemy je do zmiennej `bufor`).

`javax.microedition.media.control.RecordControl`

- `public void setRecordStream(OutputStream` *`strumien`*`)` — ustawia strumień wyjścia, do którego zapisywane są dane z mikrofonu lub innego urządzenia nagrywającego.
- `public void startRecord()` — rozpoczyna nagrywanie, o ile odtwarzacz jest w stanie `STARTED`.
- `public void stopRecord()` — przerywa nagrywanie. Powoduje przejście kontrolki nagrywającej w stan wstrzymania. Aby ponownie włączyć kontrolkę, należy wywołać metodę `startRecord()`.
- `public void commit()` — kończy nagrywanie. W przeciwieństwie do metody `stopRecord()` po wywołaniu tej metody nie można ponownie wywołać metody `startRecord()`. Wymaga to ustawienia nowego strumienia lub adresu docelowego.

- `public void setRecordLocator(String url)` — ustawia adres docelowy dla zapisywanych danych na podany w parametrze. Można stosować zamiennie z metodą `setRecordStream()`.

Odtwarzanie nagrania

Na obecnym etapie tworzenia klasy `Odtwarzacz` dysponujemy zmienną `bufor`, która zawiera nagranie. Następnym krokiem będzie dodanie do niej dwóch metod wykorzystujących tę zmienną w celu:

- odtworzenia nagrania,
- zapisu nagrania do RMS.

Pierwszą z funkcji realizuje metoda `odtworzNagranie()`:

Odtwarzacz.java

```
public void odtworzNagranie()
 {
  if (!nagrywa && bufor.length>0)
  {
   try
   {
    p = Manager.createPlayer(new ByteArrayInputStream(bufor),"audio/x-wav");
    p.realize();
    p.prefetch();
    p.start();
   }
   catch (Exception me)
   {
    me.printStackTrace();
   }
  }
 }
```

Podstawowym warunkiem wykonania tej metody jest to, aby odtwarzacz nie zapisywał właśnie żadnych danych (`nagrywa == false`). Dodatkowym zabezpieczeniem przed odtworzeniem nagrania jeszcze przed jego zarejestrowaniem (np. tuż po wyświetleniu formatki) jest warunek `bufor.length>0`. Proces odtwarzania jest realizowany według standardowego schematu. Warto jednak zwrócić uwagę na parametry metody `create ↪Player()`. Tablica jest ponownie przekształcana na strumień; jednocześnie wymuszamy standardowy typ audio, czyli `audio/x-wav`.

Treść drugiej z metod jest znacznie krótsza, gdyż większość implementacji została ujęta w klasie `Czytnik`:

Odtwarzacz.java

```
public void zapisz(String nazwa) throws Exception
{
 if (!nagrywa && bufor.length>0)
 {
```

```
    czytnik.zapisz(nazwa,bufor);
    bufor = new byte[0];
   }
  }
```

W metodzie `zapisz()`zostało zastosowane identyczne zabezpieczenie jak w metodzie `odtworzNagranie()`. Metoda `zapisz()` obiektu `czytnik` zapisuje w nowym rekordzie zarówno treść nagrania, jak i typ MIME.

Obsługa aparatu

Ostatnią skomplikowaną funkcją multimedialną do zaprogramowania jest obsługa aparatu fotograficznego. Również ona składa się z dwóch etapów: zainicjalizowania działania aparatu oraz wykonania zdjęcia. Pierwsza z tych czynności do złudzenia przypomina odtwarzanie pliku wideo:

Odtwarzacz.java

```
public Item pobierajObraz()
 {
  Item obrazAparatu = null;
  try
  {
   p = Manager.createPlayer("capture://video");
   p.realize();
   aparat = (VideoControl)p.getControl("VideoControl");
   if (aparat != null)
   {
    obrazAparatu = (Item)aparat.initDisplayMode(VideoControl.
    ↪USE_GUI_PRIMITIVE,null);
   }
   p.start();
  } catch (Exception e)
  {
   e.printStackTrace();
  }
  return obrazAparatu;
 }
```

Proszę zwrócić uwagę, że powyższa metoda różni się od `odtworzNagranie()` jednym, choć istotnym elementem. Jest to adres, z którego pobierane będą dane — określamy go jako wejściowe źródło danych wideo — czyli po prostu aparat.

Odtwarzacz działa, na formatce `formaAparat` widnieje komponent klasy `Item` wyświetlający obraz z aparatu. Pozostaje udostępnić użytkownikowi możliwość zrobienia zdjęcia:

Odtwarzacz.java

```
public void pobierzZdjecie()throws Exception
{
 if (aparat != null)
 {
     new Thread(new Runnable(){
  public void run()
```

```
 {
  try
  {
   byte[] bufor_zdjecia = aparat.getSnapshot(typZdjecia);
   zapisz(typZdjeciaWTypMIME(typZdjecia),bufor_zdjecia);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }
    }).start();
  }
}
```

Na początku, jak zwykle, znajduje się zabezpieczenie — tym razem upewniamy się, że kontrolka wideo istnieje. Wykonanie zdjęcia, podobnie jak w przypadku nagrania, wymaga uzyskania zgody, dlatego treść metody znajduje się w nowym wątku. Kluczowe wywołanie to `getSnapshot()`. Zwraca ono tablicę bajtów zawierającą zdjęcie zapisane z użyciem typu podanego w parametrze metody. Następnie wykorzystujemy drugą wersję metody `zapisz()`, która zapisuje dane do RMS z podanym buforem (a nie z buforem nagrania dźwiękowego, jak to miało miejsce w przypadku pierwszej wersji tej metody).

Pomocnicza metoda `typZdjeciaWTypMIME()` przekształca nazwy typów w formacie odpowiadającym właściwości `video.snapshot.encodings` (opisanym wcześniej) na typy MIME:

Odtwarzacz.java

```
public String typZdjeciaWTypMIME(String typ)
 {
  if (typ.equals("encoding=jpeg"))
   return "image/jpeg";
  if (typ.equals("encoding=png"))
   return "image/png";
  if (typ.equals("encoding=gif"))
   return "image/gif";
  return "";
 }
```

Przerywanie odtwarzania i zamykanie odtwarzacza

W metodzie obsługi poleceń klasy MIDletu pojawiała się metoda wywoływana np. przy powrocie z formy `formaOdtwarzacz` do `menu`. Jej zadaniem było przerwanie odtwarzanego strumienia tak, aby możliwe było odtworzenie następnego. Oto jej treść:

Odtwarzacz.java

```
public void przerwij()
{
 if (p!=null && p.getState() == Player.STARTED)
 {
  try
  {
```

```
    p.stop();
    p.close();
    aparat = null;
   } catch (MediaException me)
   {
    me.printStackTrace();
   }
  }
 }
```

Przede wszystkim należy ponownie rozpatrzeć warunki przerwania działania odtwarzacza. Nie ma sensu wywoływać metody, gdy odtwarzacz nie jest zajęty odtwarzaniem, stąd warunek `p.getState() == Player.STARTED`. W bloku `try` zamykamy odtwarzacz i kontrolkę-aparat.

Ostatnią istotną metodą jest zamknięcie odtwarzacza i zwolnienie zajętych przez niego zasobów, co odbywa się przy zamykaniu całego MIDletu. Odpowiedzialna za to jest metoda `koniec()`:

Odtwarzacz.java

```
public void koniec()
{
 try
 {
  czytnik.koniec();
  if (p!=null && p.getState()!=Player.CLOSED)
  {
   p.close();
   p.deallocate();
  }
 } catch (Exception e)
 {
  e.printStackTrace();
 }
}
```

Najpierw należy zakończyć pracę podległego odtwarzaczowi obiektu — czyli obiektu `czytnik`. Następnie, jeśli odtwarzacz nie jest zamknięty, należy go zamknąć oraz bezwzględnie zwolnić jego zasoby.

W klasie `Odtwarzacz` znajdują się jeszcze cztery pomocnicze metody, a wśród nich druga wersja metody `zapisz()`:

Odtwarzacz.java

```
 public void zapisz(String nazwa, byte[] dane) throws Exception
{
 czytnik.zapisz(nazwa,dane);
}
 public String[] pobierzID()
{
 return czytnik.pobierzID();
}
public boolean jestObrazek(String nazwa)
```

```
{
 return (nazwa.startsWith("image"));
}
```

Metoda `pobierzID()` jest wykorzystywana w klasie MIDletu przy wyświetlaniu formatki `listaRms`, a `jestObrazek()` — w metodzie `przygotuj()` przy sprawdzaniu, czy dany plik jest obrazkiem (według typu MIME).

Wykorzystanie RMS w projekcie

Wszelkiego rodzaju operacje z użyciem RMS oraz innych źródeł danych, z których korzysta klasa `Odtwarzacz`, są zadeklarowane w klasie `CzytnikDanych`. Niektóre metody za pośrednictwem klasy `Odtwarzacz` wykorzystuje również klasa MIDletu. Sama instancja klasy `CzytnikDanych` jest przechowywana jedynie w klasie `Odtwarzacz`.

Lista zmiennych niniejszej klasy jest krótka; nie może jednak na niej zabraknąć klasy zbioru rekordów, czyli pola `zbior` klasy `RecordStore`:

CzytnikDanych.java

```
 private String typ;
 private InputStream strumien;
 private RecordStore zbior;
 private OdtwarzaczMIDlet m;
}
```

Oprócz zbioru rekordów i egzemplarza klasy MIDletu w kodzie znajdują się dwie bardzo ważne zmienne, które są używane w metodzie `przygotuj()` klasy `Odtwarzacz` (za pośrednictwem stosownych metod dostępu).

Do inicjalizacji zbioru dochodzi w konstruktorze:

CzytnikDanych.java

```
package pl.helion.j2mepp.odtwarzacz;

import java.io.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;
import javax.microedition.rms.*;
import javax.microedition.media.*;

public class CzytnikDanych
{
 public CzytnikDanych(OdtwarzaczMIDlet p_m)
 {
  m = p_m;
  try
  {
   zbior = RecordStore.openRecordStore("media",true);
  } catch (Exception e){}
 }
```

Jak widać, zbiór zawierający nagrania i zdjęcia nosi nazwę *media*. Pierwsza metoda, która wykorzystuje jedynie RMS, to `zapisz()`:

CzytnikDanych.java

```
public void zapisz(String nazwa, byte[] bufor)
{
 try
 {
  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  DataOutputStream out = new DataOutputStream(baos);
  out.writeUTF(nazwa);
  out.write(bufor);
  byte[] dane = baos.toByteArray();
  zbior.addRecord(dane,0,dane.length);
 } catch (Exception e)
 {
  e.printStackTrace();
 }
}
```

Na początku metody tworzymy strumienie, dzięki którym dane dowolnych typów podstawowych można przekształcić na tablicę bajtów. Następnie uzyskane dane zapisujemy jako nowy rekord. Jak widać, rekord składa się z dwóch części: nazwy typu MIME i właściwej treści danych multimedialnych.

Klasa MIDletu do wyświetlenia listy wszystkich rekordów ze zbioru wykorzystuje metodę `pobierzID()` (pośrednio poprzez metodę klasy `Odtwarzacz` o tej samej nazwie):

CzytnikDanych.java

```
public String[] pobierzID()
 {
  String[] wyniki = new String[0];
  try
  {
   wyniki = new String[zbior.getNumRecords()];
   RecordEnumeration prz = zbior.enumerateRecords(null,null,false);
   int i=0;
   while (prz.hasNextElement())
   {
    wyniki[i] = prz.nextRecordId()+"";
    i+=1;
   }
   return wyniki;
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
  return wyniki;
 }
```

Na początku tablica wyników jest inicjalizowana bez elementów. Nie można pozostawić samej deklaracji, gdyż w razie problemów z pobraniem wyników ze zbioru rekor-

dów zostałaby zwrócona wartość `null`. Proces pobierania rekordów odbywa się za pomocą obiektu interfejsu `RecordEnumeration`, aby nie „zgubić" żadnego z nich albo nie odwoływać się do identyfikatorów nieistniejących rekordów.

Nieuchronnie zbliżamy się do analizy najważniejszej metody — `analizuj()`:

CzytnikDanych.java

```
public Player analizuj(String uri)
{
  Player p = null;
  try
  {
    String protokol = uri.substring(0,4);
    if (protokol.equals("http"))
    {
 HttpConnection pol = (HttpConnection)Connector.open(uri);
 typ = this.rozszerzenieWTyp(uri);
 if (typ.startsWith("image"))
    strumien = pol.openInputStream();
 else
    p = Manager.createPlayer(uri);
   }
   if (protokol.equals("file"))
   {
 String v = System.getProperty("microedition.io.file.FileConnection.version" );
 if (v!=null)
 {
    typ = this.rozszerzenieWTyp(uri);
    if (typ.startsWith("image"))
    {
 FileConnection ic = (FileConnection)Connector.open(uri,Connector.READ);
 strumien = ic.openInputStream();
    } else
 p = Manager.createPlayer(uri);
 }
   }
   if (protokol.equals(" rms"))
   {
 String ID = uri.substring(uri.lastIndexOf('/')+1,uri.length());
 byte[] bufor = zbior.getRecord(Integer.parseInt(ID));
 DataInputStream in = new DataInputStream(new ByteArrayInputStream(bufor));
 typ = in.readUTF();
 int dlugosc = bufor.length - typ.length();
 byte[] dane = new byte[dlugosc];
 in.read(dane);
 strumien = new ByteArrayInputStream(dane);
 if (!typ.startsWith("image"))
   p = Manager.createPlayer(strumien, typ);
    }
 } catch (Exception e)
 {
      e.printStackTrace();
      m.destroyApp(true);
      m.notifyDestroyed();
```

```
    }
    return p;
  }
```

Jest to najbardziej rozbudowana z metod. Pierwszym ważnym krokiem jest ustalenie protokołu. Ponieważ nazwy obydwu standardowo używanych protokołów (`http`, `file`) składają się z czterech liter, również nazwa stworzonego przez nas protokołu (`"rmsp"`) została tak wybrana, aby uprościć kod analizujący URL.

Najłatwiej jest pobrać dane z internetu. Wystarczy utworzyć połączenie HTTP, sprawdzić typ MIME (dla obrazków zawsze zaczyna się on od słowa `image`) i udostępnić strumień **lub** utworzyć obiekt odtwarzacza.

Równie proste jest wczytywanie danych z pliku (protokół `file`). Ponownie mechanizm strumieni użyty jest do odczytu obrazków, dla pozostałych typów danych wykorzystywany jest odtwarzacz.

Ostatnia część metody `analizuj()` pobiera dane z RMS. Po pobraniu identyfikatora rekordu z URL i odczytaniu właściwego rekordu sprawdzamy typ MIME (linijka nr 106). Jedynym problemem pozostaje znalezienie długości danych — w tym celu wystarczy jednak odjąć od całej długości danych długość nazwy typu MIME. Następnie tworzymy tablicę zawierającą jedynie treść multimedialną (`dane`) i przekształcamy ją w strumień wejścia. Analogicznie jak w dwóch poprzednich przypadkach od rodzaju danych uzależniamy utworzenie odtwarzacza.

Jeśli w trakcie wczytywania danych wystąpi jakikolwiek błąd, aplikacja musi zakończyć działanie.

W metodzie `analizuj()` pojawiła się metoda konwertująca rozszerzenie pliku na przypuszczalny typ MIME:

CzytnikDanych.java

```
public String rozszerzenieWTyp(String uri)
 {
  String roz = uri.substring(uri.lastIndexOf('.')+1,uri.length());
  String typ = "";
  if (roz.equals("wav"))
   typ = "audio/x-wav";
  if (roz.equals("mpg"))
   typ = "video/mpeg";
  if (roz.equals("mid"))
   typ = "audio/midi";
  if (roz.equals("jpg"))
   typ = "image/jpeg";
  if (roz.equals("png"))
   typ = "image/png";
  if (roz.equals("gif"))
   typ = "image/gif";
  return typ;
 }
```

Na zakończenie działania aplikacji wywoływana jest metoda `koniec()`, zamykająca zbiór rekordów:

CzytnikDanych.java

```
public void koniec()
 {
  try
  {
   zbior.closeRecordStore();
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }
```

W tym celu wykorzystywana jest poznana w rozdziale 3. metoda `closeRecordStore()`.

Ostatnie dwie metody nie zawierają nic odkrywczego, są to klasyczne metody dostępu:

CzytnikDanych.java

```
  public InputStream getStrumien()
 {
  return strumien;
 }
 public String getTyp()
 {
  return typ;
 }
```

Podsumowanie

Niniejszy MIDlet należy do najbardziej rozbudowanych projektów opisanych w tej książce. Mimo zastosowania różnych interesujących metod wiele klas i interfejsów nie zostało omówionych z powodu niezmiernie dużej ich liczby, zwłaszcza w MMAPI. Pozostawia to jednak duże pole do popisu, zwłaszcza jeśli chodzi o kontrolę procesu odtwarzania za pomocą kontrolek, takich jak `PitchControl`, `RateControl`, `TempoControl`, `MIDIControl`. Tak jak i w poprzednich projektach nie obsługujemy wszystkich możliwych wyjątków, które mogą się pojawić w czasie działania aplikacji — jest to kolejny temat (wbrew pozorom niebanalny), którym dociekliwy czytelnik powinien się zająć.

Kolejny rozdział tej książki zawiera opis API związanego ze szczególnym rodzajem programów, dzięki którym MIDlety i Java zagościły na stałe w świadomości użytkowników telefonów komórkowych — czyli z grami. Co za tym idzie, w następnym, a zarazem przedostatnim rozdziale „Praktycznych projektów" napiszemy razem grę — prostą platformówkę.

J2ME

Praktyczne projekty

Wydanie II

J2ME, czyli Java 2 Micro Edition, to uproszczona wersja platformy Java, opracowana przez firmę Sun Microsystems specjalnie na potrzeby programowania urządzeń przenośnych, takich jak telefony komórkowe czy palmtopy. Umożliwia tworzenie ciekawych i wydajnych aplikacji mobilnych, które bez większych problemów można uruchamiać na sprzęcie o stosunkowo słabych parametrach technicznych. Pozwala to osobom zainteresowanym produkcją gier, programów multimedialnych czy narzędzi sieciowych swobodnie rozwinąć skrzydła w tej dziedzinie.

„J2ME. Praktyczne projekty. Wydanie II" to praktyczny przewodnik po zaawansowanych zagadnieniach związanych z tworzeniem różnego rodzaju aplikacji mobilnych przy użyciu środowiska Java. Autor pokrótce przedstawia w nim podstawowe informacje na temat projektowania i kodowania programów działających na urządzeniach przenośnych, aby szybko przejść do konkretnych przykładów zastosowania zdobytej wiedzy. Dzięki nim nauczysz się tworzyć gry, aplikacje komunikacyjne, programy multimedialne i narzędzia GPS. Jeśli chcesz szybko opanować J2ME, tej książki nie może zabraknąć na Twojej półce!

- Instalacja środowiska programisty J2ME
- Podstawowe informacje o platformie i sposobach jej używania
- Obsługa zaawansowanych wyświetlaczy
- Tworzenie aplikacji sieciowych i komunikacyjnych
- Przetwarzanie i wykorzystywanie danych XML
- Tworzenie aplikacji multimedialnych i obsługa kamer
- Projektowanie i programowanie gier
- Tworzenie aplikacji GPS

Dołącz do elitarnego grona programistów aplikacji mobilnych!

Nr katalogowy: 5796

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

helion.pl
księgarnia internetowa

Cena 39,90 zł

ISBN 978-83-246-2835-3

9 788324 628353

Informatyka w najlepszym wydaniu