PRENTICE HALL

JAVA®

Podstawy

WYDANIE X

CAY S. HORSTMANN

Helion

Tytuł oryginału: Core Java Volume I – Fundamentals (10th Edition)

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-283-2480-0

Authorized translation from the English language edition, entitled CORE JAVA VOLUME I – FUNDAMENTALS, Tenth Edition; ISBN 0134177304; by Cay S. Horstmann; published by Pearson Education, Inc,
publishing as Prentice Hall.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/javp10.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/javp10
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

13

Przygotowywanie apletów i aplikacji do użytku

W tym rozdziale:

W tej chwili powinniśmy swobodnie posługiwać się większością funkcji języka Java. Mamy też solidne podstawy programowania interfejsów graficznych. Skoro potrafimy tworzyć aplikacje użytkowe, musimy poznać techniki przygotowywania ich do użytku na komputerze użytkownika. W kwestii tej wybór często pada na **aplety** (ang. *applet*), które były powodem ogromnego zainteresowania Javą na początku jej istnienia. Aplet to specjalny rodzaj programu w Javie, który może zostać pobrany przez przeglądarkę z internetu i uruchomiony. Miał on uwolnić użytkowników od problemów związanych z instalacją oprogramowania, które byłoby pobierane na dowolne urządzenie lub komputer obsługujący Javę i podłączony do internetu.

Aplety nie spełniły pokładanych w nich oczekiwań z wielu powodów. Dlatego rozdział ten zaczynamy od technik pakowania aplikacji. Następnie przedstawiamy sposoby zapisywania przez aplikacje informacji konfiguracyjnych i preferencji użytkownika. Dodatkowo opisujemy techniki wczytywania wtyczek do aplikacji za pomocą klasy `ServiceLoader`.

Później podpowiadamy, co trzeba wiedzieć na temat apletów, na wypadek gdyby ktoś musiał jeden utworzyć albo obsługiwać. Opisujemy też mechanizm **Java Web Start**, będący alternatywnym rozwiązaniem w zakresie dostarczania aplikacji za pomocą internetu, który pod wieloma względami przypomina aplety, tylko lepiej nadaje się do wdrażania aplikacji działających poza stroną internetową.

13.1. Pliki JAR

Pakowanie aplikacji ma na celu utworzenie jednego pliku do wykorzystania przez użytkownika zamiast całej struktury katalogów pełnych plików klas. Do tego celu służą poddawane kompresji ZIP pliki Java Archive (JAR). Mogą one zawierać nie tylko pliki klas, ale również obrazy i pliki dźwiękowe.

W Javie dostępny jest też alternatywny algorytm kompresji o nazwie pack200, który został zoptymalizowany pod kątem bardziej efektywnego zmniejszania rozmiarów plików klas w porównaniu do zwykłego algorytmu ZIP. Według zapewnień firmy Oracle współczynnik kompresji plików klas sięga aż 90%. Więcej informacji na ten temat znajduje się pod adresem *http://docs.oracle.com/javase/1.5.0/docs/guide/deployment/deployment-guide/pack200.html*.

13.1.1. Tworzenie plików JAR

Do tworzenia plików JAR służy narzędzie o nazwie `jar` (w standardowej instalacji JDK znajduje się w katalogu *jdk/bin*). Najczęściej stosowane polecenie tworzące plik JAR ma następującą składnię:

```
jar cvf JARNazwaPliku Plik1 Plik2 . . .
```

Na przykład:

```
jar cvf CalculatorClasses.jar *.class icon.gif
```

Ogólny format polecenia `jar` jest następujący:

```
jar opcje Plik1 Plik2 . . .
```

Tabela 13.1 przedstawia wszystkie opcje narzędzia `jar`. Są one podobne do opcji polecenia `tar` w systemie Unix.

Tabela 13.1. Opcje narzędzia jar

Opcja	Opis
`c`	Tworzy puste archiwum i dodaje do niego pliki. Katalogi są przetwarzane rekurencyjnie.
`C`	Zmienia tymczasowo lokalizację. Na przykład polecenie `cvf JARFileName.jar -C classes *.class` przechodzi do katalogu *classes* w celu dodania klas.
`e`	Tworzy punkt startowy w manifeście (zobacz podrozdział 13.1.3, „Wykonywalne pliki JAR”).
`f`	Określa plik JAR o danej nazwie jako drugi argument wiersza poleceń. Jeśli tego parametru brakuje, `jar` wyśle wynik do standardowego wyjścia (przy tworzeniu pliku JAR) lub wczyta go ze standardowego wejścia (przy rozpakowywaniu lub tabulacji pliku JAR).
`i`	Tworzy plik indeksowy (przyspieszający wyszukiwanie w dużych archiwach).
`m`	Dodaje **manifest** do pliku JAR. Manifest jest opisem zawartości i pochodzenia pliku archiwum. Każde archiwum ma domyślny manifest, ale można utworzyć własny, który uwierzytelnia zawartość archiwum.

Tabela 13.1. Opcje narzędzia jar (ciąg dalszy)

Opcja	Opis
M	Blokuje tworzenie domyślnego pliku manifestu.
t	Wyświetla spis treści.
u	Aktualizuje istniejący plik JAR.
v	Generuje obszerne dane wyjściowe.
x	Wypakowuje pliki. Jeśli podanych zostanie kilka nazw plików, zostaną wypakowane tylko one. W przeciwnym przypadku program wypakuje wszystkie pliki.
0	Wyłącza kompresję ZIP.

W plikach JAR można pakować aplikacje, komponenty programów (tak zwane beany, o których mowa w rozdziale 11. drugiego tomu) i biblioteki kodu. Na przykład biblioteka wykonawcza JDK jest zawarta w bardzo dużym pliku o nazwie *rt.jar*.

13.1.2. Manifest

Poza klasami, obrazami i innymi plikami źródłowymi każdy plik JAR zawiera plik **manifestu**, który określa specjalne własności archiwum.

Wspomniany plik manifestu ma nazwę *MANIFEST.MF*, a jego lokalizacja to specjalny podkatalog *META-INF* w pliku JAR. Minimalna zawartość takiego pliku nie jest zbyt interesująca:

```
Manifest-Version: 1.0
```

Złożone pliki tego typu mogą zawierać znacznie więcej wpisów pogrupowanych w sekcjach. Pierwsza sekcja nosi nazwę **sekcji głównej** (ang. *main section*) i ma zastosowanie do całego pliku JAR. Kolejne sekcje określają własności różnych elementów mających nazwy, jak konkretne pliki, pakiety czy adresy URL. Każda z nich musi się zaczynać od słowa `Name`. Sekcje są rozdzielane pustą linią. Na przykład:

```
Manifest-Version: 1.0
opis całego archiwum

Name: Woozle.class
opis jednego pliku

Name: com/mycompany/mypkg/
opis pakietu
```

Aby zmienić zawartość pliku manifestu, należy dokonać niezbędnych zmian i wydać poniższe polecenie:

```
jar cfm NazwaPlikuJAR NazwaPlikuManifest . . .
```

Na przykład poniższe polecenie tworzy nowy plik JAR z plikiem manifestu:

```
jar cfm MyArchive.jar manifest.mf com/mycompany/mypkg/*.class
```

Aby zaktualizować plik manifestu istniejącego pliku JAR, należy umieścić w pliku tekstowym wpisy, które mają być dodane, i wydać następujące polecenie:

```
jar ufm MyArchive.jar manifest-additions.mf
```

Więcej informacji na temat plików JAR i manifestu można znaleźć na stronie *http://docs.oracle.com/javase/8/docs/technotes/guides/jar*.

13.1.3. Wykonywalne pliki JAR

Istnieje możliwość określenia **punktu startowego** programu, czyli klasy, od której zaczyna się działanie programu, za pomocą opcji `e` narzędzia `jar`:

```
jar cvfe MyProgram.jar com.mycompany.mypkg.MainAppClass pliki do dodania
```

Klasę główną programu można też określić w manifeście. W tym celu należy do niego dodać instrukcję o następującej postaci:

```
Main-Class: com.mycompany.mypkg.MainAppClass
```

Nie dodawaj rozszerzenia *.class* do nazwy klasy głównej.

Na końcu ostatniego wiersza w pliku manifestu musi się znajdować znak nowego wiersza. W przeciwnym przypadku plik zostanie odczytany nieprawidłowo. Błąd polegający na utworzeniu pliku tekstowego zawierającego tylko wiersz `Main-Class` bez znaku końca wiersza jest często spotykany.

W obu wymienionych przypadkach program można uruchomić przy użyciu następującego polecenia:

```
java -jar MyProgram.jar
```

W zależności od konfiguracji systemu operacyjnego może być możliwe uruchomienie aplikacji za pomocą dwukrotnego kliknięcia pliku JAR. Poniżej znajduje się opis zachowania różnych systemów w takiej sytuacji:

- W systemie Windows instalator aplikacji Java tworzy dowiązanie dla plików o rozszerzeniu *.jar*, które uruchamia te pliki za pomocą polecenia `javaw -jar` (polecenie `javaw`, w przeciwieństwie do `java`, nie otwiera okna wiersza poleceń).
- System Solaris rozpoznaje „magiczną liczbę” pliku JAR i uruchamia ją za pomocą polecenia `java -jar`.
- System Mac OS X rozpoznaje rozszerzenie *.jar* i uruchamia programy w Javie w wyniku dwukrotnego kliknięcia pliku JAR.

Jednak programy Javy w plikach JAR to nie to samo co aplikacje rodzime. W systemie Windows można skorzystać z narzędzi innych producentów służących do zamieniania plików JAR na pliki wykonywalne tego systemu. Plik JAR jest opakowywany w plik o rozszerzeniu *.exe*, który lokalizuje i uruchamia maszynę wirtualną Javy (JVM) lub informuje użytkownika, co powinien zrobić, jeśli JVM nie ma. Istnieje kilka komercyjnych i darmowych narzędzi tego typu, np.: Launch4J (*http://launch4j.sourceforge.net*) i IzPack (*http://izpack.org*).

W komputerach z systemem Mac OS X sytuacja wygląda nieco lepiej. W skład środowiska programistycznego XCode wchodzi narzędzie o nazwie Jar Bundler służące do zamieniania plików JAR na aplikacje Mac.

13.1.4. Zasoby

Klasy używane zarówno w apletach, jak i aplikacjach często wykorzystują pliki danych tego samego typu:

- pliki obrazów i zawierające dźwięk;
- pliki tekstowe zawierające łańcuchy komunikatów i etykiety przycisków;
- pliki z danymi binarnymi, na przykład opisującymi rozkład mapy.

W Javie takie pliki nazywane są **zasobami** (ang. *resources*).

W systemie Windows termin „zasób” (ang. *resource*) ma węższe znaczenie. Zasoby w tym systemie także mogą być plikami obrazów, etykietami przycisków itd., ale są związane z plikami wykonywalnymi z dostępem za pośrednictwem standardowego interfejsu programistycznego. Natomiast pliki zasobów Javy są przechowywane osobno, a nie jako części plików klas. Dostęp do zasobów i ich interpretacja zależy od programu.

Weźmy na przykład klasę o nazwie `AboutPanel`, która wyświetla komunikat widoczny na rysunku 13.1.

Rysunek 13.1. Wyświetlanie zasobu z pliku JAR

Wiadomo, że tytuł i rok wydania zostaną zmienione w kolejnym wydaniu książki. Aby ułatwić zmianę, ten tekst należy umieścić w pliku tekstowym, a nie bezpośrednio w kodzie programu.

Powstaje jednak pytanie, gdzie umieścić taki plik jak *about.txt*. Oczywiście najlepiej byłoby, aby znajdował się on razem z pozostałymi plikami programu w pliku JAR.

Program ładujący klasy potrafi znaleźć pliki klas, jeśli znajdują się gdzieś na ścieżce klas, w archiwum lub na serwerze sieciowym. Mechanizm zasobów oferuje podobną funkcjonalność dla plików, które nie są klasami. Poniżej znajduje się spis wymaganych czynności:

1. Utwórz obiekt `Class` klasy, która ma zasób, na przykład *AboutPanel.class*.
2. Jeśli zasobem jest obraz lub plik audio, wywołaj metodę `getResource(filename)` w celu uzyskania lokalizacji zasobu w postaci adresu URL. Następnie odczytaj go za pomocą metody `getImage` lub `getAudioClip`.
3. W przypadku innych zasobów niż obrazy i pliki audio dane z pliku należy wczytywać za pomocą metody `getResourceAsStream`.

Chodzi o to, aby program ładujący klasy potrafił znaleźć klasę i odszukać związane z nią zasoby w tej samej lokalizacji.

Na przykład poniższy fragment kodu tworzy ikonę z pliku *about.gif*:

```
URL url = ResourceTest.class.getResource("about.gif");
Image img = new ImageIcon(url).getImage();
```

Powyższy kod można odczytać następująco: „znajdź plik *about.gif* w tej samej lokalizacji, w której znajduje się klasa `ResourceTest`”.

Poniższe instrukcje wczytują plik *about.txt*:

```
InputStream stream = ResourceTest.class.getResourceAsStream("about.txt");
Scanner in = new Scanner(stream, "UTF-8");
```

Plik zasobu nie musi się znajdować w tym samym katalogu co klasa — może być w jakimś podkatalogu. Można zastosować hierarchiczną nazwę zasobu, jak poniższa:

```
data/text/about.txt
```

Jest to względna nazwa zasobu. Jest ona interpretowana względem pakietu klasy, która ładuje dany zasób. Należy pamiętać, że zawsze trzeba używać separatora /, bez względu na separator katalogów stosowany w systemie, w którym są przechowywane pliki zasobów. Na przykład w systemie plików systemu Windows separatory / są automatycznie zamieniane na \.

Nazwa zasobu zaczynająca się od znaku / jest bezwzględną nazwą zasobu. Jest ona lokalizowana w taki sam sposób jak klasa wewnątrz pakietu. Na przykład zasób:

```
/corejava/title.txt
```

znajduje się w katalogu *corejava* (który może być podkatalogiem ścieżki klas wewnątrz pliku JAR lub, w przypadku apletów, na serwerze sieciowym).

Jedynym przeznaczeniem funkcji ładowania zasobów jest ładowanie plików. Nie istnieją żadne standardowe metody interpretujące zawartość pliku zasobów. Każdy program musi interpretować zawartość swoich plików zasobów na swój własny sposób.

Innym często spotykanym zastosowaniem zasobów jest międzynarodowa lokalizacja programów. W plikach zasobów przechowuje się łańcuchy, które zmieniają się w zależności od języka, czyli komunikaty i etykiety interfejsu użytkownika. Dla każdego języka tworzony jest osobny plik. **API internacjonalizacji**, które zostało opisane w rozdziale 5. drugiego tomu, udostępnia standardową metodę służącą do organizacji i dostępu do plików lokalizacyjnych.

Listing 13.1 przedstawia kod programu demonstrującego ładowanie zasobów. Poniższe polecenia kompilują go, tworzą plik JAR i uruchamiają go:

```
javac resource/ResourceTest.java
jar cvfm ResourceTest.jar resource/ResourceTest.mf resource/*.class resource/*.gif
➥resource/*.txt
java -jar ResourceTest.jar
```

Listing 13.1. resource/ResourceTest.java

```
package resource;

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;

/**
 * @version 1.41 2015-06-12
 * @author Cay Horstmann
 */
public class ResourceTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(() -> {
         JFrame frame = new ResourceTestFrame();
         frame.setTitle("ResourceTest");
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         frame.setVisible(true);
      });
   }
}

/**
 * Ramka ładująca zasoby graficzne i tekstowe
 */
class ResourceTestFrame extends JFrame
{
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 300;

   public ResourceTestFrame()
   {
      setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
      URL aboutURL = getClass().getResource("about.gif");
      Image img = new ImageIcon(aboutURL).getImage();
      setIconImage(img);

      JTextArea textArea = new JTextArea();
      InputStream stream = getClass().getResourceAsStream("about.txt");
      try (Scanner in = new Scanner(stream, "UTF-8"))
     {
        while (in.hasNext())
           textArea.append(in.nextLine() + "\n");
```

```
        }
        add(textArea);
      }
   }
```

Aby przekonać się, że program pobiera pliki zasobów z archiwum JAR, a nie bieżącego katalogu, można przenieść ten program do innego folderu.

`java.lang.Class` **1.0**

- `URL getResource(String name)` **1.1**
- `InputStream getResourceAsStream(String name)` **1.1**

  Znajduje zasób w tym samym katalogu, w którym jest umieszczona klasa, i zwraca adres URL lub strumień wejściowy, za pomocą którego można ten zasób załadować. Zwraca wartość `null`, jeśli zasób nie istnieje, dzięki czemu nie powoduje wyjątku dla błędu wejścia-wyjścia.

13.1.5. Pieczętowanie pakietów

W rozdziale 4. wspomnieliśmy o możliwości **pieczętowania** (ang. *seal*) pakietów Javy w celu uniemożliwienia dodawania do nich kolejnych klas. Może być to konieczne w przypadku używania klas, metod i pól o zasięgu pakietowym. Gdyby nie pieczętowanie, inne klasy mogłyby być umieszczane w tym samym pakiecie i dzięki temu uzyskiwać dostęp do elementów pakietowych.

Jeśli na przykład pakiet `com.mycompany.util` zostanie zapieczętowany, żadna klasa spoza tego zapieczętowanego archiwum nie może być zdefiniowana za pomocą poniższej instrukcji:

```
package com.mycompany.util;
```

W tym celu wszystkie klasy pakietu należy umieścić w pliku JAR. Domyślnie pakiety w pliku JAR nie są zapieczętowane. Można zmienić to domyślne globalne ustawienie, wstawiając wiersz

```
Sealed: true
```

w głównej sekcji pliku manifestu. Aby zapieczętować tylko wybrane pakiety, należy do pliku manifestu w pliku JAR wstawić dodatkowe sekcje:

```
Name: com/mycompany/util/
Sealed: true

Name: com/mycompany/misc/
Sealed: false
```

Aby zapieczętować pakiet, należy utworzyć plik tekstowy z instrukcjami manifestu. Następnie należy uruchomić narzędzie `jar` w zwykły sposób:

```
jar cvfm MyArchive.jar manifest.mf pliki do dodania
```

13.2. Zapisywanie preferencji użytkownika

Użytkownicy oczekują, że wszystkie dokonane przez nich ustawienia zostaną zapisane i zastosowane przy ponownym uruchamianiu aplikacji. Najpierw zajmiemy się prostą techniką opartą na zapisywaniu informacji konfiguracyjnych w plikach własności, które były kiedyś stosowane w Javie. Następnie przejdziemy do opisu niezwykle funkcjonalnego API zarządzania preferencjami.

13.2.1. Słowniki własności

Słowniki własności (ang. *property map*) to struktury danych przechowujące pary klucz – wartość, które często znajdują zastosowanie jako przechowalnie danych konfiguracyjnych aplikacji. Każdy taki słownik ma trzy cechy:

- Klucze i wartości są łańcuchami.
- Można ją łatwo zapisać w pliku i załadować z niego.
- Istnieje druga tabela przechowująca wartości domyślne.

Klasa odpowiedzialna za implementację słowników własności nosi nazwę `Properties`.

Jak wiadomo, słowniki własności znajdują zastosowanie w określaniu opcji konfiguracyjnych programów. Na przykład:

```
Properties settings = new Properties();
settings.put("width", "200");
settings.put("title", "Witaj, świecie!");
```

Do zapisania takiej listy własności w pliku należy użyć metody `store`. My zapiszemy nasz słownik w pliku o nazwie *program.properties*. Drugi argument metody `store` to komentarz, który jest dodawany do pliku.

```
FileOutputStream out = new FileOutputStream("program.properties");
settings.store(out, "Ustawienia programu");
```

W pliku zostaną zapisane następujące dane:

```
#Ustawienia programu
#Mon Apr 30 07:22:52 2007
width=200
title=Witaj, świecie!
```

Do ładowania plików ustawień służą następujące instrukcje:

```
FileInputStream in = new FileInputStream("program.properties");
settings.load(in);
```

Istnieje zwyczaj przechowywania ustawień programu w podkatalogu głównego katalogu użytkownika. Nazwa tego katalogu zazwyczaj zaczyna się od kropki — w systemie Unix konwencja taka oznacza, że katalog jest katalogiem systemowym ukrytym przed użytkownikiem. W naszym przykładowym programie stosujemy się do tej konwencji.

Do sprawdzenia katalogu głównego użytkownika można wykorzystać metodę `System.getPro↪perties`, która — tak się składa — wykorzystuje obiekt typu `Properties` do zapisu danych systemowych. Katalog główny ma klucz `user.home`. Istnieje także metoda pozwalająca odczytać pojedynczy klucz:

```
String userDir = System.getProperty("user.home");
```

Dobrze jest na wszelki wypadek dostarczyć zestaw ustawień domyślnych dla programu. Klasa `Properties` dysponuje dwoma mechanizmami pozwalającymi określić ustawienia domyślne. Po pierwsze, można utworzyć łańcuch, który będzie stosowany domyślnie za każdym razem, kiedy dany klucz nie zostanie znaleziony.

```
String title = settings.getProperty("title", "Domyślny tytuł");
```

Jeśli w słowniku własności znajduje się własność `title`, parametr `title` zostanie ustawiony na jej łańcuch. W przeciwnym przypadku parametr ten przyjmie wartość `Domyślny tytuł`.

Po drugie, jeśli wpisywanie wartości domyślnej w każdym wywołaniu metody `getProperty` okaże się zbyt żmudne, wszystkie ustawienia domyślne można umieścić w drugorzędnym słowniku własności dostarczanym następnie w konstruktorze słownika głównego.

```
Properties defaultSettings = new Properties();
defaultSettings.put("width", "300");
defaultSettings.put("height", "200");
defaultSettings.put("title", "Domyślny tytuł");
. . .
Properties settings = new Properties(defaultSettings);
```

Można nawet dostarczyć domyślne ustawienia dla ustawień domyślnych. Wystarczy tylko utworzyć kolejny słownik własności i przekazać go do konstruktora `defaultSettings`. Nie jest to jednak często spotykane rozwiązanie.

Listing 13.2 przedstawia program zapisujący i ładujący ustawienia programu. Zapamiętywane są położenie, rozmiar i tytuł ramki. Wygląd programu można dostosować *według własnego uznania*, edytując plik o nazwie *.corejava/program.properties* znajdujący się w katalogu głównym.

Listing 13.2. properties/PropertiesTest.java

```
package properties;

import java.awt.EventQueue;
import java.awt.event.*;
import java.io.*;
import java.util.Properties;

import javax.swing.*;

/**
 * Program testujący mechanizm własności. Ten program zapamiętuje położenie, rozmiar i tytuł ramki.
 * @version 1.01 2015-06-16
 * @author Cay Horstmann
 */
public class PropertiesTest
{
```

```
   public static void main(String[] args)
   {
      EventQueue.invokeLater(() -> {
         PropertiesFrame frame = new PropertiesFrame();
         frame.setVisible(true);
      });
   }
}

/**
 * Ramka pobierająca dane dotyczące położenia i rozmiaru z pliku własności oraz aktualizująca ten plik
 * w momencie zamykania programu
 */
class PropertiesFrame extends JFrame
{
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 200;

   private File propertiesFile;
   private Properties settings;

   public PropertiesFrame()
   {
      // Pobranie informacji o położeniu, rozmiarze i tytule z pliku własności

      String userDir = System.getProperty("user.home");
      File propertiesDir = new File(userDir, ".corejava");
      if (!propertiesDir.exists()) propertiesDir.mkdir();
      propertiesFile = new File(propertiesDir, "program.properties");

      Properties defaultSettings = new Properties();
      defaultSettings.put("left", "0");
      defaultSettings.put("top", "0");
      defaultSettings.put("width", "" + DEFAULT_WIDTH);
      defaultSettings.put("height", "" + DEFAULT_HEIGHT);
      defaultSettings.put("title", "");

      settings = new Properties(defaultSettings);


      if (propertiesFile.exists())
        try (InputStream in = new FileInputStream(propertiesFile))
        {
          settings.load(in);
        }
        catch (IOException ex)
        {
          ex.printStackTrace();
        }

      int left = Integer.parseInt(settings.getProperty("left"));
      int top = Integer.parseInt(settings.getProperty("top"));
      int width = Integer.parseInt(settings.getProperty("width"));
      int height = Integer.parseInt(settings.getProperty("height"));
      setBounds(left, top, width, height);

      // Jeśli nie ma tytułu, użytkownik zostanie poproszony o jego podanie

      String title = settings.getProperty("title");
```

```
        if (title.equals("")) title = JOptionPane.showInputDialog("Wpisz tytuł
        ↪ramki:");
        if (title == null) title = "";
        setTitle(title);

        addWindowListener(new WindowAdapter()
           {
              public void windowClosing(WindowEvent event)
              {
                 settings.put("left", "" + getX());
                 settings.put("top", "" + getY());
                 settings.put("width", "" + getWidth());
                 settings.put("height", "" + getHeight());
                 settings.put("title", getTitle());

                 try (OutputStream out = new FileOutputStream(propertiesFile))
                 {
                   settings.store(out, "Program Properties");
                 }
                 catch (IOException ex)
                 {
                    ex.printStackTrace();
                 }
                 System.exit(0);
              }
           });
     }
}
```

Klasa `Properties` ze względów historycznych implementuje interfejs `Map<Object, ↪Object>`, a więc można używać metod `get` i `put` interfejsu `Map`. Metoda `get` zwraca obiekt typu `Object`, a metoda `put` pozwala na wstawienie do struktury dowolnego obiektu. Najlepiej cały czas posługiwać się metodami `getProperty` i `setProperty`, które działają na łańcuchach, a nie obiektach.

Obiekty typu `Properties` są zwykłymi tablicami pozbawionymi struktury hierarchicznej. Programiści często tworzą namiastkę hierarchii, odpowiednio nazywając klucze, np. `window.main.color`, `window.main.title` itd. Jednak klasa `Properties` nie zawiera żadnych metod wspomagających organizację takich hierarchii. Do przechowywania skomplikowanych informacji konfiguracyjnych lepiej używać klasy `Preferences`, która została opisana w kolejnym podrozdziale.

`java.util.Properties` **1.0**

- `Properties()`

  Tworzy pusty słownik własności.

- `Properties(Properties defaults)`

  Tworzy słownik własności z zestawem ustawień domyślnych.

  Parametr: `defaults` Wartości domyślne

- `String getProperty(String key)`

  Pobiera słownik własności. Zwraca łańcuch skojarzony z kluczem `key` lub łańcuch skojarzony z kluczem `key` w słowniku domyślnym, jeśli nie ma go w aktualnym słowniku, albo wartość `null`, jeśli nie zostanie on znaleziony także w tym drugim słowniku.

  Parametr: `key` Klucz, którego wartość ma zostać pobrana.

- `String getProperty(String key, String defaultValue)`

  Pobiera własność z domyślną wartością, jeśli klucz `key` nie zostanie znaleziony. Zwraca łańcuch skojarzony z kluczem `key` lub łańcuch domyślny, jeśli nie ma go w tablicy.

  Parametry: `key` Klucz, którego wartość ma zostać pobrana.

  `defaultValue` Wartość zwracana, jeśli dany klucz nie istnieje.

- `Object setProperty(String key, String value)`

  Parametry: `key` Klucz, z którym jest związany łańcuch do ustawienia.

  `value` Wartość, która ma zostać skojarzona z kluczem.

- `void load(InputStream in) throws IOException`

  Ładuje słownik własności ze strumienia wejściowego.

  Parametr: `in` Strumień wejściowy

- `void store(OutputStream out, String header)` **1.2**

  Zapisuje słownik własności w strumieniu wyjściowym.

  Parametry: `out` Strumień wyjściowy

  `header` Nagłówek umieszczany w pierwszym wierszu zapisywanego pliku

`java.lang.System` **1.0**

- `Properties getProperties()`

  Pobiera wszystkie właściwości systemowe. Jeśli aplikacja nie ma uprawnień do pobierania wszystkich właściwości systemowych, generowany jest wyjątek zabezpieczeń.

- `String getProperty(String key)`

  Pobiera właściwość systemową opatrzoną podanym kluczem. Jeśli aplikacja nie ma uprawnień do pobierania tej właściwości systemowej, generowany jest wyjątek zabezpieczeń. Poniższe właściwości można zawsze pobierać:

  ```
  java.version
  java.vendor
  java.vendor.url
  java.class.version
  os.name
  ```

```
os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
```

Nazwy wolno dostępnych właściwości systemowych można znaleźć w pliku *security/*
↪*java.policy* znajdującym się w katalogu środowiska uruchomieniowego Javy.

13.2.2. API Preferences

Klasa `Properties`, mimo iż umożliwia zapisywanie i odczytywanie danych konfiguracyjnych w prosty sposób, ma kilka wad:

- Nie zawsze mamy możliwość zapisania plików konfiguracyjnych w katalogu głównym użytkownika, ponieważ niektóre systemy operacyjne nie znają koncepcji katalogu głównego.
- Nie istnieje standardowa konwencja nazywania plików konfiguracyjnych, co wiąże się z ryzykiem wystąpienia konfliktów nazw, jeśli użytkownik zainstaluje kilka aplikacji Java.

Niektóre systemy operacyjne mają centralne repozytorium, w którym przechowują dane konfiguracyjne. Najlepszym przykładem takiego repozytorium jest rejestr w systemie Microsoft Windows. Klasa `Preferences` pozwala utworzyć takie repozytorium niezależnie od platformy. W systemie Windows klasa ta wykorzystuje rejestr. W systemie Linux informacje te są zapisywane w lokalnym systemie plików. Oczywiście implementacja repozytorium nie ma tajemnic dla programisty używającego klasy `Preferences`.

Repozytorium `Preferences` ma strukturę drzewiastą z nazwami ścieżek do węzłów typu */com/mycompany/myapp*. Podobnie jak w przypadku pakietów, konfliktów nazw ścieżek unika się, stosując odwrócone nazwy domen. Projektanci tego API zalecają nawet, aby ścieżkom do węzłów konfiguracyjnych nadawać takie same nazwy jak pakietom używanym w programie.

Każdy węzeł w repozytorium ma oddzielną tablicę par klucz – wartość, w której można przechowywać łańcuchy, liczby i tablice bajtów. Nie ma możliwości zapisywania obiektów serializowanych, ponieważ projektanci uznali, że format ten jest zbyt ulotny do długoterminowego przechowywania. Oczywiście ci, którzy się z tym nie zgadzają, mogą zapisywać serializowane obiekty w tablicach bajtów.

Większą elastyczność zapewniają dodatkowe równoległe drzewa. Każdy użytkownik programu ma własne drzewo i dodatkowe drzewo systemowe, które przechowuje ustawienia wspólne wszystkich użytkowników. Odpowiednie drzewo ustawień jest pobierane przez klasę `Preferences` przy użyciu pojęcia bieżącego użytkownika, rozumianego zgodnie z systemem operacyjnym.

Aby uzyskać dostęp do węzła drzewa, należy zacząć od użytkownika root lub katalogu systemowego root:

```
Preferences root = Preferences.userRoot();
```

lub

```
Preferences root = Preferences.systemRoot();
```

Następnie uzyskujemy dostęp do węzła. Można ograniczyć się do podania jego ścieżki:

```
Preferences node = root.node("/com/mycompany/myapp");
```

Za pomocą wygodnego skrótu można pobrać węzeł, którego ścieżka jest taka sama jak nazwa pakietu klasy. Wystarczy pobrać obiekt tej klasy i zastosować poniższe wywołanie:

```
Preferences node = Preferences.userNodeForPackage(obj.getClass());
```

lub

```
Preferences node = Preferences.systemNodeForPackage(obj.getClass());
```

Zazwyczaj `obj` jest referencją `this`.

Mając węzeł, można uzyskać dostęp do jego tablicy par klucz – wartość za pomocą następujących metod:

```
String get(String key, String defval)
int getInt(String key, int defval)
long getLong(String key, long defval)
float getFloat(String key, float defval)
double getDouble(String key, double defval)
boolean getBoolean(String key, boolean defval)
byte[] getByteArray(String key, byte[] defval)
```

Należy pamiętać, że przy odczytywaniu informacji trzeba dostarczyć wartość domyślną, na wypadek gdyby w repozytorium brakowało jakichś danych. Wartości domyślne są wymagane z kilku powodów. Danych może brakować, ponieważ użytkownik nigdy nie ustawił danej opcji. Niektóre ograniczone platformy mogą nie mieć repozytorium, a urządzenia przenośne mogą być tymczasowo odłączone od repozytorium.

Do zapisu danych w repozytorium służą metody `put`:

```
put(String key, String value)
putInt(String key, int value)
```

i tak dalej.

Listę wszystkich kluczy zapisanych w węźle można uzyskać za pomocą metody `String[] keys`. Nie ma jednak obecnie sposobu na sprawdzenie typu wartości określonego klucza.

Centralne repozytoria, takie jak rejestr w systemie Windows, zazwyczaj cierpią z dwóch powodów:

- Z czasem zamieniają się w śmietnisko pełne przestarzałych informacji.
- Dane konfiguracyjne plączą się w repozytorium, przez co trudno jest je przenieść na inną platformę.

Klasa Preferences ma rozwiązanie drugiego problemu. Można wyeksportować ustawienia poddrzewa (lub — rzadziej — jednego węzła) za pomocą poniższych metod:

```
void exportSubtree(OutputStream out)
void exportNode(OutputStream out)
```

Dane są zapisywane w formacie XML. Można je zaimportować do innego repozytorium za pomocą następującego wywołania:

```
void importPreferences(InputStream in)
```

Poniżej znajduje się zawartość przykładowego pliku:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE preferences SYSTEM "http://java.sun.com/dtd/preferences.dtd">
<preferences EXTERNAL_XML_VERSION="1.0">
  <root type="user">
    <map/>
    <node name="com">
      <map/>
      <node name="horstmann">
        <map/>
        <node name="corejava">
          <map>
            <entry key="left" value="11"/>
            <entry key="top" value="9"/>
            <entry key="width" value="453"/>
            <entry key="height" value="365"/>
            <entry key="title" value="Witaj, Świecie!"/>
          </map>
        </node>
      </node>
    </node>
  </root>
</preferences>
```

Jeśli program wykorzystuje klasę Preferences, należy umożliwić użytkownikowi import i eksport ustawień, co ułatwia przenoszenie ustawień na inny komputer. Program na listingu 13.3 demonstruje omówioną technikę. Zapisuje on położenie, rozmiar i tytuł głównego okna. Po zamknięciu i ponownym uruchomieniu programu okno będzie wyglądało dokładnie tak samo jak przed zamknięciem programu.

Listing 13.3. preferences/PreferencesTest.java

```
package preferences;

import java.awt.*;
import java.io.*;
import java.util.prefs.*;

import javax.swing.*;
import javax.swing.filechooser.*;

/**
 * Program testujący ustawianie preferencji. Zapamiętuje położenie, rozmiar i tytuł ramki.
 * @version 1.03 2015-06-12
 * @author Cay Horstmann
 */
```

```
public class PreferencesTest
{
   public static void main(String[] args)
   {
      EventQueue.invokeLater(() -> {
         PreferencesFrame frame = new PreferencesFrame();
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         frame.setVisible(true);
      });
   }
}

/**
 * Ramka pobierająca dane dotyczące położenia i rozmiaru z preferencji użytkownika oraz aktualizująca
 * preferencje w momencie zamykania programu
 */
class PreferencesFrame extends JFrame
{
   private static final int DEFAULT_WIDTH = 300;
   private static final int DEFAULT_HEIGHT = 200;
   private Preferences root = Preferences.userRoot();
   private Preferences node = root.node("/com/horstmann/corejava");

   public PreferencesFrame()
   {
      // Sprawdzanie położenia, rozmiaru i tytułu w preferencjach

      int left = node.getInt("left", 0);
      int top = node.getInt("top", 0);
      int width = node.getInt("width", DEFAULT_WIDTH);
      int height = node.getInt("height", DEFAULT_HEIGHT);
      setBounds(left, top, width, height);

      // Jeśli nie ma tytułu, użytkownik zostanie poproszony o jego podanie

      String title = node.get("title", "");
      if (title.equals(""))
         title = JOptionPane.showInputDialog("Podaj tytuł ramki:");
      if (title == null) title = "";
      setTitle(title);

      // Utworzenie okna wyboru plików wyświetlającego pliki XML

      final JFileChooser chooser = new JFileChooser();
      chooser.setCurrentDirectory(new File("."));
      chooser.setFileFilter(new FileNameExtensionFilter("Pliki XML", "xml"));

      // utworzenie menu

      JMenuBar menuBar = new JMenuBar();
      setJMenuBar(menuBar);
      JMenu menu = new JMenu("Plik");
      menuBar.add(menu);

      JMenuItem exportItem = new JMenuItem("Eksportuj preferencje");
      menu.add(exportItem);
      exportItem
            .addActionListener(event -> {
```

```
            if (chooser.showSaveDialog(PreferencesFrame.this) ==
            ↪JFileChooser.APPROVE_OPTION)
            {
               try
               {
                  savePreferences();
                  OutputStream out = new FileOutputStream(chooser
                        .getSelectedFile());
                  node.exportSubtree(out);
                  out.close();
               }
               catch (Exception e)
               {
                  e.printStackTrace();
               }
            }
         });

      JMenuItem importItem = new JMenuItem("Importuj preferencje");
      menu.add(importItem);
      importItem
            .addActionListener(event -> {
               if (chooser.showOpenDialog(PreferencesFrame.this) ==
               ↪JFileChooser.APPROVE_OPTION)
               {
                  try
                  {
                     InputStream in = new FileInputStream(chooser
                           .getSelectedFile());
                     Preferences.importPreferences(in);
                     in.close();
                  }
                  catch (Exception e)
                  {
                     e.printStackTrace();
                  }
               }
            });

      JMenuItem exitItem = new JMenuItem("Zamknij");
      menu.add(exitItem);
      exitItem.addActionListener(event -> {
         savePreferences();
         System.exit(0);
      });
   }

   public void savePreferences()
   {
      node.putInt("left", getX());
      node.putInt("top", getY());
      node.putInt("width", getWidth());
      node.putInt("height", getHeight());
      node.put("title", getTitle());
   }
}
```

`java.util.prefs.Preferences` **1.4**

- `Preferences userRoot()`

  Zwraca węzeł preferencji root użytkownika programu.

- `Preferences systemRoot()`

  Zwraca węzeł preferencji root systemu.

- `Preferences node(String path)`

  Zwraca węzeł, do którego można uzyskać dostęp z bieżącego węzła za pośrednictwem podanej ścieżki `path`. Jeśli ścieżka jest bezwzględna (czyli zaczyna się od znaku /), szukanie węzła zaczyna się od korzenia drzewa zawierającego ten węzeł preferencji. Jeśli węzeł w podanej ścieżce nie istnieje, zostanie utworzony.

- `Preferences userNodeForPackage(Class cl)`
- `Preferences systemNodeForPackage(Class cl)`

  Zwraca węzeł w bieżącym drzewie użytkownika lub drzewo systemowe, którego ścieżka bezwzględna węzła odpowiada nazwie pakietu klasy `cl`.

- `String[] keys()`

  Zwraca wszystkie klucze należące do węzła.

- `String get(String key, String defval)`
- `int getInt(String key, int defval)`
- `long getLong(String key, long defval)`
- `float getFloat(String key, float defval)`
- `double getDouble(String key, double defval)`
- `boolean getBoolean(String key, boolean defval)`
- `byte[] getByteArray(String key, byte[] defval)`

  Zwraca wartość skojarzoną z danym kluczem lub podaną wartość domyślną, jeśli z kluczem nie jest skojarzona żadna wartość lub skojarzona wartość jest nieprawidłowego typu, lub magazyn preferencji nie jest dostępny.

- `void put(String key, String value)`
- `void putInt(String key, int value)`
- `void putLong(String key, long value)`
- `void putFloat(String key, float value)`
- `void putDouble(String key, double value)`
- `void putBoolean(String key, boolean value)`
- `void putByteArray(String key, byte[] value)`

  Zapisuje parę klucz – wartość w węźle.

- `void exportSubtree(OutputStream out)`

  Zapisuje preferencje węzła i jego potomków w określonym strumieniu.

- `void exportNode(OutputStream out)`

  Zapisuje preferencje węzła (ale nie jego potomków) w określonym strumieniu.

- `void importPreferences(InputStream in)`

  Importuje preferencje zawarte w określonym strumieniu.

13.3. Moduły ładowania usług

Niektóre aplikacje specjalnie buduje się tak, by było łatwo rozbudowywać ich funkcjonalność za pomocą wtyczek. Istnieją nawet platformy ułatwiające to zadanie, np. OSGi (*http://osgi.org*), które wykorzystuje się w środowiskach programistycznych, serwerach aplikacji i innych złożonych programach. Opis tych platform wykracza daleko poza temat tej książki, ale warto wiedzieć, że JDK również zawiera prosty mechanizm do ładowania wtyczek. Jego opis zamieszczamy poniżej.

Twórcy programów obsługujących wtyczki często pozostawiają programistom tych dodatków pewien zakres wolności pod względem sposobu implementacji swoich produktów. Dobrze jest też mieć do wyboru kilka implementacji. Klasa `ServiceLoader` sprawia, że ładowanie wtyczek spełniających wymagania określonego wspólnego interfejsu jest bardzo łatwe.

Należy zdefiniować interfejs (lub, jeśli ktoś woli, nadklasę) z metodami, które muszą być zdefiniowane przez każdy egzemplarz usługi. Powiedzmy na przykład, że stworzyliśmy usługę szyfrowania:

```
package serviceLoader;

public interface Cipher
{
  byte[] encrypt(byte[] source, byte[] key);
  byte[] decrypt(byte[] source, byte[] key);
  int strength();
}
```

Dostawca usług zapewnia przynajmniej jedną klasę implementującą tę usługę, np.:

```
package serviceLoader.impl;

public class CaesarCipher implements Cipher
{
  public byte[] encrypt(byte[] source, byte[] key)
  {
    byte[] result = new byte[source.length];
    for (int i = 0; i < source.length; i++)
      result[i] = (byte)(source[i] + key[0]);
    return result;
  }
```

```
  public byte[] decrypt(byte[] source, byte[] key)
  {
    return encrypt(source, new byte[] { (byte) -key[0] });
  }

  public int strength() { return 1; }
}
```

Klasy implementacyjne mogą należeć do dowolnego pakietu, tzn. nie muszą się znajdować w tym samym pakiecie co interfejs usługi. Każda z nich musi natomiast mieć konstruktor bezargumentowy.

Teraz dodamy nazwy klas do pliku tekstowego UTF-8 znajdującego się w katalogu *META-INF/services*. Nazwa tego pliku musi się zgadzać z pełną nazwą klasy. W naszym przypadku oznacza to, że plik *META-INF/services/serviceLoader.Cipher* będzie zawierał następujący wiersz:

```
serviceLoader.impl.CaesarCipher
```

Teraz dostarczyliśmy tylko jedną klasę implementacyjną, ale w razie potrzeby możemy dodać ich więcej i wybierać tę, która w danym przypadku jest najodpowiedniejsza.

Po tych przygotowaniach program inicjalizuje moduł ładujący usługi w następujący sposób:

```
public static ServiceLoader<Cipher> cipherLoader = ServiceLoader.load(Cipher.class);
```

Wystarczy to zrobić tylko raz w programie.

Metoda `iterator` modułu ładowania usług zwraca iterator do przeglądania wszystkich dostarczonych implementacji usługi. (Szerzej na temat iteratorów piszemy w rozdziale 9.). Aby je przejrzeć, najłatwiej posłużyć się rozszerzoną pętlą `for`, w której można wybrać odpowiedni obiekt do świadczenia usługi.

```
public static Cipher getCipher(int minStrength)
{
  for (Cipher cipher : cipherLoader) // niejawnie wywołuje cipherLoader.iterator()
  {
    if (cipher.strength() >= minStrength) return cipher;
  }
  return null;
}
```

`java.util.ServiceLoader<S>` **1.6**

- `static <S> ServiceLoader<S> load(Class<S> service)`

  Tworzy moduł ładowania usług służący do wczytywania klas implementujących określony interfejs usługowy.

- `Iterator<S> iterator()`

  Zwraca iterator, który leniwie ładuje klasy usług, tzn. załadowanie klasy następuje po przesunięciu iteratora do przodu.

13.4. Aplety

Aplety to programy w języku Java dołączane do stron HTML. Strona HTML musi poinformować przeglądarkę, które aplety ma załadować oraz gdzie mają one być rozmieszczone. Jak nietrudno się domyślić, znacznik służący do wstawiania apletów musi dostarczać informacje dotyczące lokalizacji plików klas oraz samego apletu (jego rozmiaru, lokalizacji itd.). Przeglądarka pobiera pliki klas z internetu (lub katalogu na urządzeniu użytkownika) i automatycznie uruchamia aplet.

Na początku istnienia apletów jedyną przeglądarką, która je obsługiwała, była HotJava firmy Sun. Oczywiście znalazło się niewiele osób, które były skłonne używać oddzielnej przeglądarki dla jednej dodatkowej funkcji. Aplety zyskały prawdziwą popularność z chwilą dołączenia przez firmę Netscape maszyny wirtualnej Javy do przeglądarki Navigator. Niedługo później to samo zrobiła firma Microsoft w przeglądarce Internet Explorer. Niestety Microsoft podciął skrzydła Netscape, opornie dodając obsługę starych wersji Javy w Internet Explorerze, aż w końcu całkiem tego zaniechał.

Problem ten rozwiązuje narzędzie Java Plug-in. Integruje się ono z przeglądarkami jako rozszerzenie, co umożliwia uruchamianie apletów przy wykorzystaniu zewnętrznego środowiska uruchomieniowego Javy.

Przez wiele lat było to dobre rozwiązanie, a jako aplety powszechnie implementowało się materiały edukacyjne, aplikacje biznesowe oraz niektóre gry. Niestety firma Sun, a po jej zniknięciu firma Oracle niespiesznie łatały luki w zabezpieczeniach maszyny wirtualnej Javy, które od czasu do czasu ktoś wykrywał i wykorzystywał w niecnych zamiarach. Słabo zabezpieczona maszyna wirtualna narażała użytkowników na spore ryzyko, więc producenci przeglądarek postanowili ograniczyć możliwość korzystania z Javy w swoich produktach. Niektórzy zezwalali na używanie tylko najnowszej wersji wtyczki Java, a inni porzucili wsparcie dla architektury wtyczkowej. Niestety reakcja firmy Oracle na to, co się dzieje, była rozczarowująca. Firma zaczęła wymagać, aby wszystkie aplety były podpisane cyfrowo (zobacz sekcję 13.4.9, „Podpisywanie kodu").

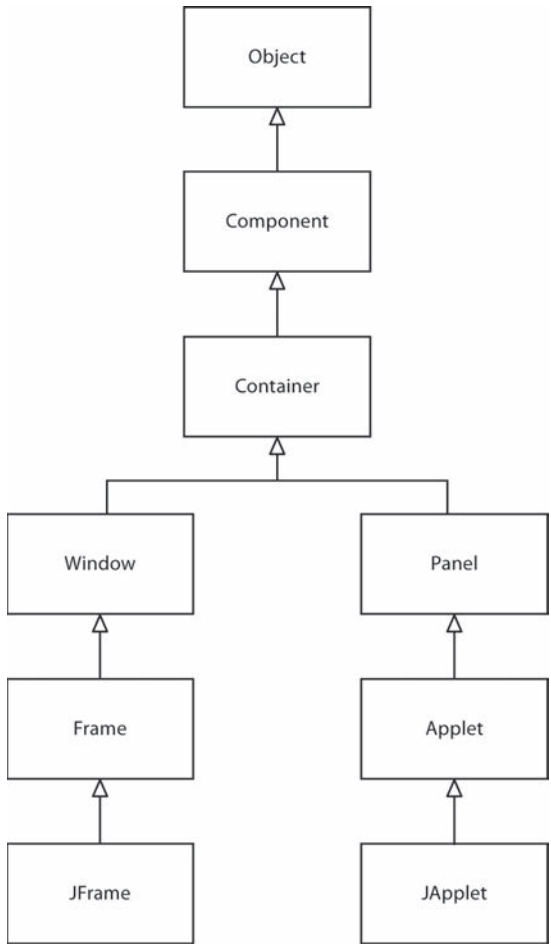
Z tych powodów dziś wdrożenie apletu Java jest sporym wyzwaniem dla programisty, a dla użytkownika — jego uruchomienie. Dlatego też dalsza część tego rozdziału zainteresuje głownie tych czytelników, którzy spotykają się w pracy ze starymi apletami.

Aby móc uruchamiać aplety opisywane w tym rozdziale, należy zainstalować najnowszą wersję narzędzia Java Plug-in i upewnić się, że przeglądarka jest połączona z wtyczką. Dodatkowo, aby móc testować aplety, należy tak skonfigurować wtyczkę, by ufała plikom lokalnym. Piszemy o tym w podrozdziale 2.5, „Tworzenie i uruchamianie apletów".

13.4.1. Prosty aplet

Aby tradycji stało się zadość, przerobimy program *NotHelloWorld* na aplet. Aplet to zwykła klasa Javy rozszerzająca klasę `java.applet.Applet`. Do implementacji apletów użyjemy Swinga. Wszystkie nasze aplety będą rozszerzały klasę `JApplet`, która jest nadklasą apletów Swing. Jak widać na rysunku 13.2, klasa `JApplet` jest bezpośrednią podklasą klasy `Applet`.

Rysunek 13.2. Diagram dziedziczenia klasy Applet



Jeśli aplet zawiera składniki Swing, należy rozszerzyć klasę `JApplet`. Komponenty Swing w zwykłym kontenerze `Applet` nie są poprawnie rysowane.

Na listingu 13.4 przedstawiona jest apletowa wersja programu „nie powitalnego”.

Zwróć uwagę na duże podobieństwo do wersji z rozdziału 10. Ponieważ aplet działa w oknie przeglądarki, nie trzeba było definiować metody zamykającej.

Listing 13.4. applet/NotHelloWorld.java

```
package applet;

import java.awt.*;
import javax.swing.*;

/**
 * @version 1.24 2015-06-12
 * @author Cay Horstmann
 */
public class NotHelloWorld extends JApplet
{
   public void init()
   {
      EventQueue.invokeLater(() -> {
         JLabel label = new JLabel("To nie jest aplet Witaj, świecie",
            SwingConstants.CENTER);
         add(label);
      });
   }
}
```

Uruchomienie powyższego apletu wymaga wykonania trzech czynności:

1. Kompilacji plików źródłowych na pliki klas.
2. Spakowania klas do pliku JAR (zobacz sekcję 13.1.1, „Tworzenie plików JAR”).
3. Utworzenia pliku HTML zawierającego informacje o lokalizacji plików klas oraz rozmiarze apletu.

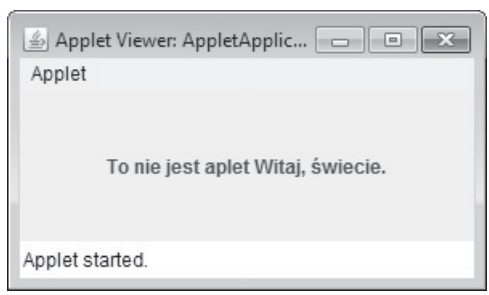
Poniżej znajduje się jego zawartość:

```
<applet code="applet/NotHelloWorld.class" width="300" height="300">
</applet>
```

Dobrym pomysłem jest przetestowanie apletu we wchodzącej w skład pakietu JDK **przeglądarce apletów** (ang. *applet viewer*) przed otwarciem go w przeglądarce internetowej. Poniższe polecenie wiersza poleceń otwiera nasz aplet we wspomnianej przeglądarce:

```
appletviewer NotHelloWorldApplet.html
```

Argumentem narzędzia `appletviewer` jest nazwa pliku HTML, nie pliku klasy. Rysunek 13.3 przedstawia nasz aplet w przeglądarce apletów.

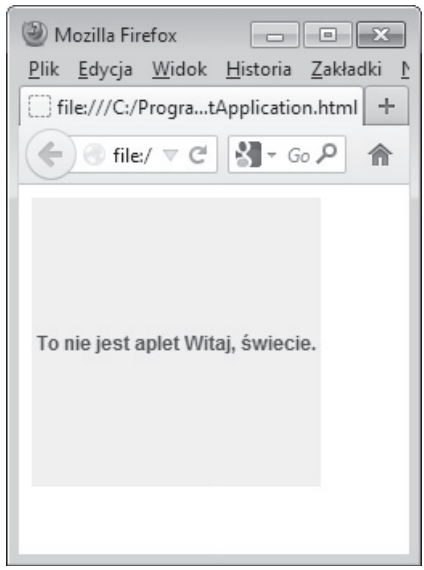


Rysunek 13.3. Aplet w przeglądarce apletów

Przeglądarka apletów dobrze sprawdza się jako pierwszy etap testowania. Trzeba jednak w końcu uruchomić aplet w przeglądarce, aby sprawdzić, jak będzie się prezentował użytkownikowi. Przeglądarka apletów pokazuje sam aplet bez otaczającego go kodu HTML. Jeśli strona HTML zawiera kilka znaczników `applet`, przeglądarka otworzy kilka okien.

Aby obejrzeć swój aplet w przeglądarce, wystarczy załadować w niej odpowiednią stronę HTML (rysunek 13.4). Jeśli aplet nie pojawia się, należy zainstalować narzędzie Java Plug-in i zezwolić mu na ładowanie niepodpisanych apletów lokalnych (zobacz podrozdział 2.5, „Tworzenie i uruchamianie apletów”).

Rysunek 13.4. Aplet w przeglądarce



Jeśli w aplecie zostaną wprowadzone jakieś zmiany i zostanie on raz jeszcze poddany kompilacji, konieczne jest ponowne uruchomienie przeglądarki, aby załadowała nowe pliki klas. Samo odświeżenie strony nie spowoduje załadowania nowej wersji apletu. Bywa to kłopotliwe przy szukaniu błędów. Można uniknąć ponownego uruchamiania przeglądarki dzięki użyciu **konsoli Javy**. Należy uruchomić tę konsolę i wydać polecenie `x`, które czyści pamięć programu ładującego klasy. Wtedy po odświeżeniu strony zostanie załadowana nowa wersja apletu. W systemie Windows należy otworzyć panel kontrolny Java Plug-in znajdujący się w *Panelu sterowania*. W systemie Linux należy użyć polecenia `jcontrol` i zażądać wyświetlenia panelu kontrolnego Javy. Konsola będzie się pojawiać za każdym razem, kiedy ładowany jest aplet.

Graficzne aplikacje w Javie można z łatwością przekonwertować na aplety. Cały kod dotyczący interfejsu użytkownika pozostaje bez zmian. Oto lista niezbędnych czynności:

1. Utwórz stronę HTML z odpowiednim znacznikiem wstawiającym aplet.
2. Utwórz podklasę klasy `JApplet`. Zdefiniuj tę klasę jako publiczną. W przeciwnym razie apletu nie będzie można załadować.
3. Usuń z aplikacji metodę `main`. Nie twórz ramki dla aplikacji, ponieważ będzie ona wyświetlana w oknie przeglądarki.

4. Przenieś kod inicjalizujący z konstruktora ramki do metody `init` apletu. Nie trzeba jawnie konstruować obiektu apletu — przeglądarka robi to automatycznie i wywołuje metodę `init`.
5. Usuń wywołanie metody `setSize`. W apletach za rozmiary odpowiadają parametry HTML `width` i `height`.
6. Usuń wywołanie metody `setDefaultCloseOperation`. Apletu nie można zamknąć — jego działanie kończy się w chwili zamknięcia przeglądarki.
7. Jeśli w programie znajduje się wywołanie metody `setTitle`, należy je usunąć. Aplety nie mają pasków tytułu (można oczywiście nadać tytuł samej stronie HTML za pomocą znacznika `title`).
8. Nie wywołuj metody `setVisible(true)`. Aplet jest wyświetlany automatycznie.

`java.applet.Applet` **1.0**

- `void init()`

  Jest wywoływana przy pierwszym ładowaniu apletu. Metodę tę należy przesłonić i umieścić w niej cały kod inicjalizujący.

- `void start()`

  Należy przesłonić tę metodę i umieścić w niej kod, który ma być wykonywany *za każdym razem*, gdy użytkownik odwiedza stronę zawierającą ten aplet. Do typowych działań należy tu reaktywacja wątku.

- `void stop()`

  Należy przesłonić tę metodę i umieścić w niej kod, który ma być wykonywany *za każdym razem*, gdy użytkownik opuszcza stronę zawierającą ten aplet. Do typowych działań należy tu dezaktywacja wątku.

- `void destroy()`

  Metodę tę należy przedefiniować, wstawiając do niej kod wykonywany w momencie zamknięcia przeglądarki.

- `void resize(int width, int height)`

  Wymusza zmianę rozmiaru apletu. Byłaby to doskonała metoda, gdyby działała na stronach internetowych. Niestety obecnie nie działa w przeglądarkach, ponieważ zakłóca ich mechanizm rozkładu elementów na stronie.

13.4.2. Znacznik applet i jego atrybuty

Znacznik `applet` w najprostszej postaci może wyglądać następująco:

```
<applet code="NotHelloWorld.class" archive="NotHelloWorld.jar" width="300"
height="100"></applet>
```

Znacznik `applet` ma następujące atrybuty:

- `width`, `height`

  Atrybuty te są wymagane i określają szerokość i wysokość apletu w pikselach. W przeglądarce apletów wymiary te są traktowane jako początkowe, ale rozmiar każdego okna tej przeglądarki można zmienić. W przeglądarce internetowej *nie ma możliwości* zmiany rozmiaru apletu. Optymalny rozmiar apletu, tak aby wyglądał dobrze u każdego użytkownika, należy określić metodą prób i błędów.

- `align`

  Określa wyrównanie apletu. Wartości tego atrybutu są takie same jak atrybutu `align` znacznika `img`.

- `vspace`, `hspace`

  Określają liczbę pikseli nad i pod apletem (`vspace`) oraz po jego obu stronach (`hspace`).

- `code`

  Określa nazwę pliku klasy apletu.

  Ścieżka musi się zgadzać z pakietem, do którego należy klasa. Jeśli na przykład klasa apletu należy do pakietu `com.mycompany`, atrybut wygląda następująco `code="com/mycompany/MyApplet.class"`. Można też stosować alternatywny zapis w postaci `code="com.mycompany.MyApplet.class"`.

  Atrybut `code` określa tylko nazwę klasy apletu. Oczywiście sam aplet może zawierać także inne pliki klas. Kiedy przeglądarka załaduje klasę zawierającą aplet, zorientuje się, że potrzebne są dodatkowe klasy, i je również załaduje.

- `archive`

  Określa listę plików JAR lub plików zawierających klasy i inne zasoby apletu, które są pobierane z serwera przed jego załadowaniem. Pliki JAR na liście są rozdzielane przecinkami:

  ```
  <applet code="MyApplet.class"
    archive="MyClasses.jar,corejava/CoreJavaClasses.jar"
    width="100" height="150">
  ```

- `codebase`

  Określa adres URL, pod którym znajdują się pliki JAR (a kiedyś także pliki klas) do załadowania.

- `object`

  Przestarzały atrybut określający nazwę pliku zawierającego poddany *serializacji* obiekt apletu, który miał być wykorzystany w celu przechowania stanu apletu. Funkcja ta jest już bezużyteczna, ponieważ nie ma możliwości podpisania pliku poddanego serializacji.

- `alt`

  W atrybucie tym można zdefiniować tekst do wyświetlenia, gdy przeglądarka nie obsługuje Javy.

Jeśli przeglądarka w ogóle nie rozpoznaje apletów, czyli pochodzi z czasów prehistorycznych, ignoruje znaczniki `applet` i `param`. W takiej sytuacji zostanie wyświetlony tekst znajdujący się pomiędzy znacznikami `<applet>` i `</applet>`. Natomiast przeglądarki obsługujące aplety nie wyświetlają tego tekstu. Na przykład:

```
<applet . . . alt="Gdyby Twoja przeglądarka obsługiwała Javę, w tym miejscu
➥byłby widoczny mój aplet.">
  Gdyby Twoja przeglądarka obsługiwała Javę, w tym miejscu byłby widoczny mój aplet.
</applet>
```

- `name`

Twórcy skryptów wykorzystują ten atrybut do odwoływania się do apletu w swoich skryptach. Przeglądarki Netscape i Internet Explorer zezwalają na wywoływanie metod apletów na stronie za pośrednictwem JavaScriptu.

Aby uzyskać dostęp do apletu z poziomu JavaScriptu, najpierw należy nadać mu nazwę:

```
<applet ... name="mine"></applet>
```

Dzięki temu można się do niego odwoływać za pomocą zapisu `document.applets.`*`nazwaapletu`*. Na przykład:

```
var myApplet = document.applets.mine;
```

Można wywoływać metody apletu:

```
myApplet.init();
```

Atrybut `name` ma także kluczowe znaczenie w sytuacjach, kiedy dwa aplety znajdujące się na tej samej stronie mają się ze sobą bezpośrednio komunikować. Należy nadać nazwę każdemu apletowi. Łańcuch ten należy przekazać do metody `getApplet` z klasy `AppletContext`. Mechanizm ten, o nazwie **komunikacja między apletami** (ang. *inter-applet communication*), został opisany nieco dalej w tym rozdziale.

Na stronie *http://www.javaworld.com/javatips/jw-javatip80.html* Francis Lu wykorzystuje mechanizm komunikacji Javy z JavaScriptem do rozwiązania odwiecznego problemu zmiany rozmiaru apletu, na stałe ustawionego przez atrybuty `width` i `height`. Jest to dobry przykład integracji tych dwóch języków.

13.4.3. Parametry przekazujące informacje do apletów

Podobnie jak aplikacje wykorzystują informacje podawane za pośrednictwem wiersza poleceń, aplety używają parametrów podawanych w plikach HTML. Służy do tego znacznik `param` i jego atrybuty. Wyobraźmy sobie, że chcemy, aby na stronie internetowej można było ustawić krój czcionki używanej w aplecie. Można do tego użyć następującego kodu HTML:

```
<applet code="FontParamApplet.class" width="200" height="200">
  <param name="font" value="Helvetica"/>
</applet>
```

Następnie wartość parametru pobieramy za pomocą metody `getParameter` z klasy `Applet`:

```
public class FontParamApplet extends JApplet
{
  public void init()
  {
    String fontName = getParameter("font");
    . . .
  }
  . . .
}
```

Metodę `getParameter` można wywoływać tylko w metodzie `init` apletu, nie w konstruktorze, ponieważ w chwili jego wykonywania parametry nie są jeszcze gotowe. Ponieważ układ większości bardziej rozbudowanych apletów jest zdeterminowany przez parametry, zalecamy zaniechanie używania konstruktorów w apletach. Cały kod inicjalizujący można umieścić w metodzie `init`.

Parametry są zawsze zwracane jako łańcuchy. Jeśli wymagana jest liczba, łańcuch trzeba przekonwertować na typ liczbowy. Służą do tego standardowe metody, takie jak `parseInt` z klasy `Integer`.

Na przykład kod HTML zawierający parametr `size`, który określa rozmiar czcionki, mógłby wyglądać następująco:

```
<applet code="FontParamApplet.class" ...>
  <param name="font" value="Helvetica"/>
  <param name="size" value="24"/>
</applet>
```

Poniższy fragment kodu demonstruje sposób odczytu parametru liczbowego:

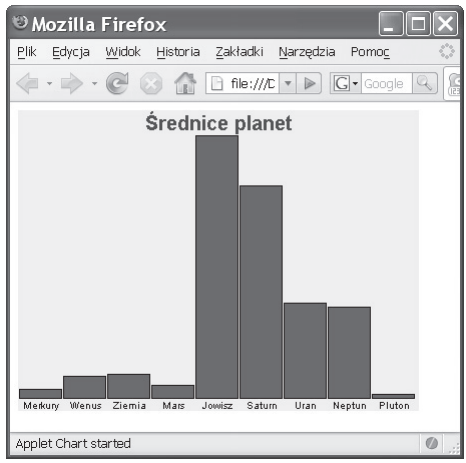
```
public class FontParamApplet extends JApplet
{
  public void init()
  {
    String fontName = getParameter("font");
    int fontSize = Integer.parseInt(getParameter("size"));
    . . .
  }
}
```

Przy porównywaniu wartości atrybutu `name` ze znacznika `param` z argumentem metody `getParameter` nie jest rozpoznawana wielkość liter.

Poza upewnieniem się, że parametry w kodzie pasują, należy sprawdzić, czy parametr `size` został ustawiony, czy nie. Służy do tego prosty test na obecność wartości `null`. Na przykład:

```
int fontsize;
String sizeString = getParameter("size");
if (sizeString == null) fontSize = 12;
else fontSize = Integer.parseInt(sizeString);
```

Rysunek 13.5 przedstawia aplet rysujący wykres słupkowy, który w dużym stopniu wykorzystuje parametry.



Rysunek 13.5. Aplet wyświetlający wykres

Aplet ten pobiera etykiety i wysokości słupków z wartości atrybutów znacznika `param`. Poniżej znajduje się kod HTML strony widocznej na rysunku 13.5.

```
<applet code="Chart.class" width="400" height="300">
  <param name="title" value="Średnice planet"/>
  <param name="values" value="9"/>
  <param name="name.1" value="Merkury"/>
  <param name="name.2" value="Wenus"/>
  <param name="name.3" value="Ziemia"/>
  <param name="name.4" value="Mars"/>
  <param name="name.5" value="Jowisz"/>
  <param name="name.6" value="Saturn"/>
  <param name="name.7" value="Uran"/>
  <param name="name.8" value="Neptun"/>
  <param name="name.9" value="Pluton"/>
  <param name="value.1" value="3100"/>
  <param name="value.2" value="7500"/>
  <param name="value.3" value="8000"/>
  <param name="value.4" value="4200"/>
  <param name="value.5" value="88000"/>
  <param name="value.6" value="71000"/>
  <param name="value.7" value="32000"/>
  <param name="value.8" value="30600"/>
  <param name="value.9" value="1430"/>
</applet>
```

Można było utworzyć w aplecie tablicę łańcuchów i tablicę liczb, ale wykorzystanie mechanizmu parametrów ma dwie zalety. Po pierwsze, na jednej stronie można wyświetlić kilka kopii tego samego apletu, pokazujących różne wykresy — trzeba zastosować kilka znaczników `applet` z różnymi zestawami parametrów. Po drugie, można zmieniać dane przedstawione na wykresie. Wprawdzie średnice planet jeszcze przez jakiś czas się nie zmienią, ale wyobraźmy sobie, że na stronie przedstawiamy tygodniowy wykres sprzedaży. Stronę internetową łatwo się aktualizuje, ponieważ jest to czysty tekst. Edytowanie i kompilowanie plików Javy wymaga znacznie więcej wysiłku.

Istnieją nawet komercyjne komponenty JavaBean (tak zwane beany), które tworzą o wiele atrakcyjniejsze wykresy. Podając parametry takiemu zakupionemu komponentowi, nie trzeba nawet wiedzieć nic na temat tworzenia wykresów.

Listing 13.5 przedstawia kod źródłowy omawianego apletu rysującego wykres. Należy zauważyć, że metoda `init` pobiera parametry, a metoda `paintComponent` rysuje wykres.

Listing 13.5. chart/Chart.java

```
package chart;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * @version 1.34 2015-06-12
 * @author Cay Horstmann
 */
public class Chart extends JApplet
{
   public void init()
   {
      EventQueue.invokeLater(() -> {
         String v = getParameter("values");
         if (v == null) return;
         int n = Integer.parseInt(v);
         double[] values = new double[n];
         String[] names = new String[n];
         for (int i = 0; i < n; i++)
         {
            values[i] = Double.parseDouble(getParameter("value." + (i + 1)));
            names[i] = getParameter("name." + (i + 1));
         }

         add(new ChartComponent(values, names, getParameter("title")));
      });
   }
}

/**
 * Komponent rysujący wykres słupkowy.
 */
class ChartComponent extends JComponent
{
   private double[] values;
   private String[] names;
   private String title;

   /**
    * Tworzy obiekt typu ChartComponent.
    * @param v tablica wartości wykresu
    * @param n tablica nazw wartości
    * @param t tytuł wykresu
    */
   public ChartComponent(double[] v, String[] n, String t)
   {
```

```
      values = v;
      names = n;
      title = t;
   }

   public void paintComponent(Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;

      // Obliczanie wartości minimalnej i maksymalnej.
      if (values == null) return;
      double minValue = 0;
      double maxValue = 0;
      for (double v : values)
      {
         if (minValue > v) minValue = v;
         if (maxValue < v) maxValue = v;
      }
      if (maxValue == minValue) return;

      int panelWidth = getWidth();
      int panelHeight = getHeight();

      Font titleFont = new Font("SansSerif", Font.BOLD, 20);
      Font labelFont = new Font("SansSerif", Font.PLAIN, 10);

      // Obliczanie szerokości tytułu.
      FontRenderContext context = g2.getFontRenderContext();
      Rectangle2D titleBounds = titleFont.getStringBounds(title, context);
      double titleWidth = titleBounds.getWidth();
      double top = titleBounds.getHeight();

      // Rysowanie tytułu.
      double y = -titleBounds.getY();   // wysokość
      double x = (panelWidth - titleWidth) / 2;
      g2.setFont(titleFont);
      g2.drawString(title, (float) x, (float) y);

      // Obliczanie szerokości etykiet słupków.
      LineMetrics labelMetrics = labelFont.getLineMetrics("", context);
      double bottom = labelMetrics.getHeight();

      y = panelHeight - labelMetrics.getDescent();
      g2.setFont(labelFont);

      // Obliczanie współczynnika skali i szerokości słupków.
      double scale = (panelHeight - top - bottom) / (maxValue - minValue);
      int barWidth = panelWidth / values.length;

      // Rysowanie słupków.
      for (int i = 0; i < values.length; i++)
      {
         // Uzyskanie współrzędnych prostokąta tworzącego słupek.
         double x1 = i * barWidth + 1;
         double y1 = top;
         double height = values[i] * scale;
         if (values[i] >= 0) y1 += (maxValue - values[i]) * scale;
         else
         {
```

```
            y1 += maxValue * scale;
            height = -height;
         }

         // Wypełnienie słupka i rysowanie jego obrysu.
         Rectangle2D rect = new Rectangle2D.Double(x1, y1, barWidth - 2, height);
         g2.setPaint(Color.RED);
         g2.fill(rect);
         g2.setPaint(Color.BLACK);
         g2.draw(rect);

         // Rysowanie etykiety na środku pod słupkiem.
         Rectangle2D labelBounds = labelFont.getStringBounds(names[i], context);

         double labelWidth = labelBounds.getWidth();
         x = x1 + (barWidth - labelWidth) / 2;
         g2.drawString(names[i], (float) x, (float) y);
      }
   }
}
```

`java.applet.Applet` **1.0**

- `public String getParameter(String name)`

  Pobiera wartość parametru zdefiniowanego w znaczniku `param` na stronie internetowej ładującej aplet. W łańcuchu `name` rozpoznawane są małe i wielkie litery.

- `public String getAppletInfo()`

  Tę metodę wielu programistów przedefiniowuje, aby zwracała łańcuch zawierający informacje o autorze, wersji i prawach autorskich do apletu. Informacje te należy podawać poprzez przesłonięcie tej metody w klasie apletu.

- `public String[][] getParameterInfo()`

  Tę metodę można przedefiniować, aby zwracała tablicę opcji znacznika `param`, obsługiwanych przez aplet. Każdy wiersz zawiera trzy pozycje: nazwę, typ i opis parametru. Na przykład:

  ```
  "fps", "1-10", "ramek na sekundę"
  "repeat", "boolean", "powtórzyć pętlę obrazu?"
  "images", "url", "katalog zawierający obrazy"
  ```

13.4.4. Dostęp do obrazów i plików audio

W apletach można używać obrazów i plików audio. W chwili pisania tego tekstu obsługiwane były następujące formaty plików graficznych: GIF, PNG i JPEG, oraz plików audio: AU, AIFF, WAV i MIDI. Animowane gify są także poprawnie obsługiwane.

Lokalizację obrazów i plików audio określa się za pomocą względnych adresów URL. Bazowy adres URL zazwyczaj sprawdza się za pomocą metody `getDocumentBase` lub `getCodeBase`. Pierwsza z wymienionych metod pobiera adres URL strony HTML zawierającej aplet, a druga adres URL katalogu bazowego kodu.

Bazowy adres URL i lokalizację pliku należy przekazać do metody `getImage` lub `getAudioClip`. Na przykład:

```
Image cat = getImage(getCodeBase(), "images/cat.gif");
AudioClip meow = getAudioClip(getCodeBase(), "audio/meow.au");
```

Wyświetlania obrazów nauczyliśmy się w rozdziale 10. Jeśli chodzi o pliki audio, wystarczy wywołać metodę `play`. Metodę `play` z klasy `Applet` można nawet wywołać bez uprzedniego załadowania pliku audio.

```
play(getCodeBase(), "audio/meow.au");
```

`java.applet.Applet` **1.0**

- `URL getDocumentBase()`

  Pobiera adres URL strony zawierającej aplet.

- `URL getCodeBase()`

  Pobiera adres URL katalogu bazowego z kodem, z którego ładowany jest aplet. Jest to albo bezwzględny adres URL katalogu wskazywanego przez atrybut `codebase`, albo adres pliku HTML, jeśli atrybut `codebase` nie istnieje.

- `void play(URL url)`
- `void play(URL url, String name)`

  Pierwsza wersja odtwarza plik dźwiękowy znajdujący się pod podanym adresem URL. Druga tworzy ścieżkę względną wobec podanego adresu URL z podanego łańcucha. Jeśli pliku audio nie ma, nic się nie dzieje.

- `AudioClip getAudioClip(URL url)`
- `AudioClip getAudioClip(URL url, String name)`

  Pierwsza wersja odtwarza plik dźwiękowy znajdujący się pod podanym adresem URL. Druga tworzy ścieżkę względną wobec podanego adresu URL z podanego łańcucha. Jeśli plik audio nie istnieje, metody te zwracają wartość `null`.

- `Image getImage(URL url)`
- `Image getImage(URL url, String name)`

  Zwraca obiekt typu `Image` zawierający obraz wskazywany przez adres URL. Jeśli obraz nie istnieje, zwracana jest natychmiast wartość `null`. W przeciwnym przypadku zostaje uruchomiony osobny wątek ładujący obraz.

13.4.5. Środowisko działania apletu

Aplety rezydują w przeglądarce internetowej lub przeglądarce apletów. Mogą one wysyłać do przeglądarek żądania dotyczące wykonania określonych działań, typu pobranie pliku audio, wyświetlenie komunikatu w pasku stanu lub otwarcie nowej strony. Przeglądarka może wykonać żądanie lub je zignorować. Jeśli na przykład aplet działający w przeglądarce apletów zgłosi żądanie otwarcia nowej strony, zostanie ono zignorowane.

Do komunikacji z przeglądarką aplet wykorzystuje metodę `getAppletContext`. Zwraca ona obiekt implementujący interfejs typu `AppletContext`. Konkretną implementację tego inter-

fejsu można traktować jako ścieżkę komunikacyjną pomiędzy apletem a przeglądarką. Poza metodami `getAudioClip` i `getImage` interfejs `AppletContext` zawiera także inne przydatne metody, które opisujemy w kilku kolejnych podrozdziałach.

13.4.6. Komunikacja pomiędzy apletami

Na jednej stronie internetowej może się znajdować kilka apletów. Jeśli wszystkie one pochodzą z jednej bazy kodowej, mogą się ze sobą komunikować. Jest to jednak zaawansowana technika, której nie używa się zbyt często.

Jeśli każdy aplet w pliku HTML ma określoną nazwę w atrybucie `name`, za pomocą metody `getApplet` z interfejsu `AppletContext` można utworzyć odwołanie do tego apletu. Jeśli na przykład plik HTML zawiera poniższy znacznik:

```
<applet code="Chart.class" width="100" height="100" name="Chart1">
```

odwołanie do apletu daje poniższa instrukcja:

```
Applet chart1 = getAppletContext().getApplet("Chart1");
```

Do czego można wykorzystać takie odwołanie? Jeśli klasa `Chart` zawiera metodę przyjmującą nowe dane i ponownie rysującą wykres, metodę tę można wywołać, wykonując odpowiednie rzutowanie.

```
((Chart) chart1).setData(3, "Ziemia", 9000);
```

Listę wszystkich apletów znajdujących się na stronie można wyświetlić bez względu na to, czy mają one atrybut `name`, czy nie. Metoda `getApplets` zwraca obiekt typu wyliczeniowego. Poniższa pętla drukuje nazwy klas wszystkich apletów znajdujących się na stronie:

```
Enumeration<Applet> e = getAppletContext().getApplets();
while (e.hasMoreElements())
{
  Applet a = e.nextElement();
  System.out.println(a.getClass().getName());
}
```

Aplety z różnych stron nie mogą się ze sobą komunikować.

13.4.7. Wyświetlanie elementów w przeglądarce

Aplety mają dostęp do dwóch miejsc w oknie przeglądarki: paska stanu i obszaru, na którym wyświetlane są strony. W obu przypadkach używa się metod z klasy `AppletContext`.

Aby wyświetlić tekst w pasku stanu, należy użyć metody `showStatus`. Na przykład:

```
showStatus("Ładowanie danych . . . proszę czekać");
```

Z naszego doświadczenia wynika, że zastosowanie metody `showStatus` jest ograniczone. Przeglądarka również używa paska stanu i w większości przypadków zastępuje dostarczony przez programistę tekst informacją typu *Applet running*. W związku z tym w pasku stanu można wyświetlać komunikaty typu *Ładowanie danych…*, ale nie takie, które są dla użytkownika ważne.

Aby zmusić przeglądarkę do otwarcia nowej strony, należy użyć metody `showDocument`, którą można wywołać na kilka sposobów. Najprostsze wywołanie polega na podaniu jako argumentu adresu URL strony, która ma zostać otwarta:

```
URL u = new URL("http://horstmann.com/index.html");
getAppletContext().showDocument(u);
```

Jednak takie wywołanie może być problematyczne, ponieważ nowa strona zastępuje aktualną, co powoduje zniknięcie apletu. Aby wrócić do apletu, konieczne jest naciśnięcie przycisku *Wstecz* w przeglądarce.

Aby nowa strona została otwarta w nowym oknie, należy metodzie `showDocument` podać dodatkowy parametr (zobacz tabelę 13.2). Łańcuch `_blank` wymusza otwarcie dokumentu w nowym oknie. Co ciekawsze, korzystając z ramek HTML, można podzielić okno na kilka części o unikatowych nazwach, w których będą wyświetlane dokumenty żądane przez aplet również znajdujący się w jednej z ramek. Przykładowy kod prezentujemy w kolejnym podrozdziale.

Przeglądarka apletów nie wyświetla stron internetowych. Metoda `showDocument` jest przez nią ignorowana.

Tabela 13.2. Metoda showDocument

Parametr określający cel	Lokalizacja
`_self` lub brak	Bieżąca ramka
`_parent`	Ramka nadrzędna
`_top`	Ramka najwyższego rzędu
`_blank`	Nowe okno najwyższego rzędu, bez nazwy
Dowolny inny łańcuch	Ramka o podanej nazwie; jeśli nie ma ramki o takiej nazwie, otwierane jest nowe okno o takiej nazwie

`java.applet.Applet` **1.2**

- `public AppletContext getAppletContext()`

  Tworzy dostęp do środowiska przeglądarki, w której działa aplet. Przy użyciu tych informacji można kontrolować większość przeglądarek.

- `void showStatus(String msg)`

  Pokazuje podany łańcuch na pasku stanu przeglądarki internetowej.

`java.applet.AppletContext` **1.0**

- `Enumeration<Applet> getApplets()`

  Zwraca wyliczenie (zobacz rozdział 9.) wszystkich apletów znajdujących się w tym samym środowisku, czyli na jednej stronie internetowej.

- `Applet getApplet(String name)`

  Zwraca aplet o podanej nazwie znajdujący się w bieżącym kontekście. Zwraca wartość `null`, jeśli aplet nie istnieje. Przeszukiwana jest tylko bieżąca strona internetowa.

- `void showDocument(URL url)`
- `void showDocument(URL url, String target)`

  Otwiera nową stronę internetową w przeglądarce. Pierwsza wersja zastępuje starą stronę nową. Druga otwiera nową stronę w miejscu określonym przez parametr `target` (tabela 13.2).

13.4.8. Piaskownica

Kiedy kod uruchamiany na komputerze jest pobierany ze zdalnego miejsca, sprawą pierwszorzędną staje się zawsze bezpieczeństwo. Wejście na stronę powoduje automatyczne uruchomienie wszystkich znajdujących się na niej apletów. Aplikację Java Web Start może uruchomić jedno kliknięcie. Gdyby kliknięcie odnośnika lub wejście na stronę internetową pozwalało na uruchomienie dowolnego kodu na komputerze użytkownika, przestępcy przeżywaliby złoty wiek wykradania poufnych informacji, danych finansowych i przejmowania komputerów użytkowników w celu rozsyłania spamu.

Technologia Java dysponuje zaawansowanym modelem ochrony, który zapobiega wykorzystywaniu jej do nikczemnych postępków. Model ten został szczegółowo opisany w tomie drugim. **Menedżer zabezpieczeń** (ang. *security manager*) kontroluje dostęp do wszystkich zasobów systemowych. Przy standardowych ustawieniach zezwala tylko na nieszkodliwe operacje. Aby możliwe było wykonywanie dodatkowych operacji, użytkownik musi wyraźnie na to zezwolić apletowi lub aplikacji.

Co pobierany zdalnie kod *może* robić na wszystkich platformach? Zawsze można wyświetlać obrazy, odtwarzać dźwięki, odpowiadać na naciśnięcia przez użytkownika klawiszy i przycisków myszki oraz wysyłać dane wprowadzone przez użytkownika do hosta, z którego został załadowany kod. Taka funkcjonalność wystarcza do zaprezentowania faktów i liczb oraz interakcji z programem edukacyjnym lub grą. Ograniczone środowisko wykonawcze jest często nazywane **piaskownicą** (ang. *sandbox*). Kod działający w piaskownicy nie może nic zmieniać w systemie użytkownika ani go szpiegować.

Programy działające w piaskownicy mają następujące ograniczenia:

- Nie mogą uruchamiać *żadnych* lokalnych programów.
- Nie mogą czytać ani zapisywać plików w lokalnym systemie plików.
- Nie mają dostępu do żadnych informacji o komputerze lokalnym, z wyjątkiem wersji Javy i kilku mało ważnych danych na temat systemu operacyjnego. Kod działający w piaskownicy w szczególności nie ma dostępu do nazwy użytkownika, adresów e-mail itd.
- Programy ładowane z serwera zdalnego nie mogą się komunikować z żadnym hostem poza tym, z którego pochodzą — serwer taki nosi nazwę **serwera pochodzenia** (ang. *originating host*). Dzięki temu użytkownik jest chroniony przed programami, które mogą wykradać wewnętrzne zasoby.

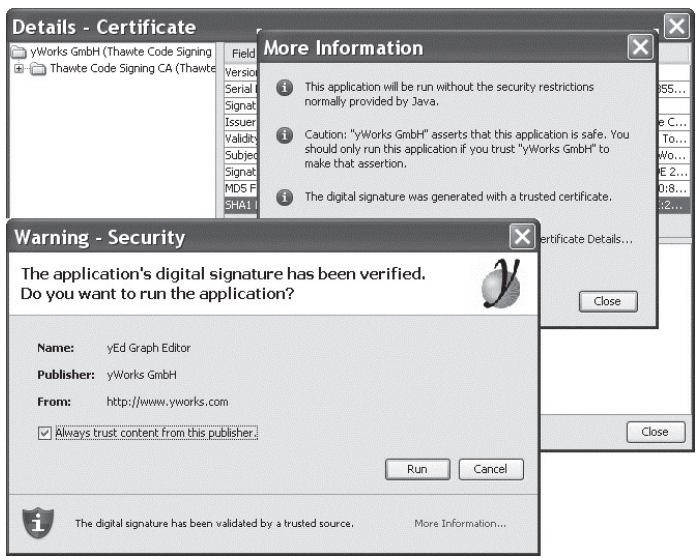
- Wszystkie wyskakujące okna zawierają komunikat ostrzegawczy. Ma on na celu zabezpieczenie przed pomyleniem ich z oknem aplikacji lokalnej. Istnieją obawy, że nic niepodejrzewający użytkownik wejdzie na stronę internetową, podstępem zostanie zmuszony do uruchomienia zdalnego kodu, a następnie wpisze hasło lub numer karty kredytowej, które zostaną przesłane z powrotem do serwera. We wczesnych wersjach JDK komunikat ten brzmiał bardzo złowieszczo: *Untrusted Java Applet Window*. W każdej kolejnej wersji brzmienie to było nieco łagodzone: *Unauthenticated Java Applet Window* czy *Warning: Java Applet Window*. Obecnie jest to drobny trójkąt, którego większość użytkowników nawet nie zauważy.

Koncepcja piaskownicy straciła już na znaczeniu. Kiedyś każdy mógł wdrażać kod do uruchamiania w tym ograniczonym środowisku i tylko kod działający poza nim musiał być cyfrowo podpisany. Obecnie podpisu cyfrowego wymagają wszystkie programy uruchamiane za pośrednictwem programu Java Plug-in, niezależnie od tego, czy działają one w piaskownicy, czy nie.

13.4.9. Podpisywanie kodu

Pliki JAR aplikacji Java Web Start muszą być **podpisane cyfrowo**. Podpisany plik JAR ma certyfikat określający tożsamość tego, kto go podpisał. Techniki kryptograficzne dają pewność, że certyfikat taki nie jest sfałszowany, a wszelkie próby zmiany jego zawartości są natychmiast wykrywane.

Wyobraźmy sobie, że pobieramy aplikację utworzoną i podpisaną cyfrowo przez firmę yWorks GmbH z certyfikatem wydanym przez ośrodek certyfikacji Thawte (rysunek 13.6). Odbierając aplikację, mamy pewność, że:



Rysunek 13.6. Certyfikat bezpieczeństwa

1. Kod aplikacji nie został zmieniony w żaden sposób od chwili jej podpisania.
2. Podpis rzeczywiście pochodzi od firmy yWorks.

3. Certyfikat naprawdę został wydany przez ośrodek Thawte (mechanizm Java Plug-in potrafi sprawdzać certyfikaty wydane przez Thawte i kilka innych instytucji; istnieje również możliwość zainstalowania innych „podstawowych certyfikatów").

Jeśli klikniemy odnośnik *More Information*, dowiemy się, że aplikacja zostanie uruchomiona bez zwyczajowych ograniczeń. To, czy zainstalować tę aplikację, czy nie, w dużej mierze zależy od tego, czy ufamy firmie yWorks.

Koszty uzyskania certyfikatu od jednego z obsługiwanych dostawców wynoszą kilkaset dolarów rocznie, a niektóre organizacje wymagają dodatkowo dowodu istnienia firmy lub licencji biznesowej. Kiedyś niektórzy programiści Javy po prostu generowali własne certyfikaty, które wykorzystywali do podpisywania kodu. Oczywiście Java Plug-in nie ma możliwości sprawdzenia, czy są one poprawne. Mimo to kiedyś program ten wyświetlał taki certyfikat użytkownikowi do zatwierdzenia. Mechanizm ten był kompletnie bezużyteczny, ponieważ tylko nieliczni użytkownicy wiedzieli, jaka jest różnica między bezpiecznym i niebezpiecznym certyfikatem. Dlatego też niebezpieczne certyfikaty nie są już obsługiwane.

Obecnie ktoś, kto chce rozprowadzić aplet Java lub aplikację Web Start, nie ma wielkiego wyboru. Musi zdobyć certyfikat od odpowiedniej organizacji uznawanej przez Java Plug-in i podpisywać przy jego użyciu swoje pliki JAR.

Wiele firm nawiązuje stałą współpracę z organizacjami certyfikującymi, więc jeśli pracujesz w jednej z nich, to wystarczy zamówić certyfikat do podpisania swojego kodu. Jeżeli nie, to warto się rozejrzeć, ponieważ ceny bywają bardzo zróżnicowane i niektórzy dostawcy oferują nieco dogodniejsze warunki odbiorcom indywidualnym.

Wraz z certyfikatem otrzymasz instrukcję, jak go zainstalować w magazynie kluczy Javy — specjalnym pliku chronionym hasłem, z którego certyfikat może zostać pobrany w trakcie procesu podpisywania. Magazyn kluczy i hasło należy przechowywać w bezpiecznym miejscu.

Następnie trzeba zdecydować, jakie uprawnienia będą potrzebne. Do wyboru są uprawnienia piaskownicy i pełne. Wyboru dokonuje się w pliku manifestu (zobacz sekcję 13.1.2, „Manifest").

Dodaj instrukcję `Permissions: sandbox` lub `Permissions: all-permissions`, np.:

```
Manifest-Version: 1.0
Permissions: all-permissions
```

Użyj narzędzia `jar`:

```
jar cvfm MyApplet.jar manifest.mf mypackage/*.class
```

Element `applet` w pliku HTML powinien mieć zdefiniowany atrybut `archive="MyApplet.jar"`.

Na koniec należy podpisać plik JAR. Oto odpowiednie polecenie:

```
jarsigner -keystore plikmagazynukluczy -tsa URLznacznikaczasu MyApplet.jar aliasklucza
```

Adres URL do pobrania znacznika czasu otrzymuje się od wydawcy certyfikatu. Alias klucza też został przydzielony przez wydawcę certyfikatu. Można go sprawdzić za pomocą poniższego polecenia:

```
keytool -keystore plikzkluczami -list
```

Ewentualnie alias klucza można zmienić za pomocą opcji `-changealias` polecenia `keytool`. (Więcej informacji na temat narzędzia `keytool` znajduje się w rozdziale 9. drugiego tomu).

Teraz pozostaje już tylko wysłanie podpisanego pliku JAR i pliku HTML z elementem `applet` na serwer.

Uprawnienia aplikacji Java można dokładnie kontrolować, o czym szerzej piszemy w rozdziale 12. drugiego tomu. System ten jednak nigdy nie był używany w sposób mający jakiekolwiek znaczenie dla zwykłych użytkowników. Program Java Plug-in udostępnia tylko dwa poziomy bezpieczeństwa — piaskownicę i wszystkie uprawnienia.

13.5. Java Web Start

Java Web Start jest technologią uruchamiania aplikacji bezpośrednio z internetu. Aplikacje Java Web Start mają następujące cechy:

- Są zazwyczaj dostarczane za pośrednictwem przeglądarki. Po pobraniu aplikacja Java Web Start może być uruchamiana bez użycia przeglądarki.
- Nie rezydują w oknie przeglądarki. Aplikacja działa we własnym oknie ramowym, poza przeglądarką.
- Nie wykorzystują implementacji Javy przeglądarki. Przeglądarka uruchamia tylko zewnętrzną aplikację, podobnie jak w przypadku innych programów, takich jak Adobe Acrobat lub Real Audio.
- Aplikacjom podpisanym cyfrowo można nadawać dowolne prawa dostępu. Niepodpisane aplikacje działają w **piaskownicy** (ang. *sandbox*), która nie zezwala na potencjalnie niebezpieczne operacje.

13.5.1. Wdrażanie aplikacji Java Web Start

Przygotowywanie aplikacji do dostarczania za pośrednictwem mechanizmu Java Web Start polega na spakowaniu jej do jednego lub większej liczby plików JAR. Następnie należy utworzyć plik deskryptora w formacie JNLP (ang. *Java Network Launch Protocol*). Pliki umieszczamy na serwerze.

Dodatkowo serwer sieciowy musi raportować typ MIME `application/x-java-jnlp-file` dla plików z rozszerzeniem *.jnlp* (typ MIME umożliwia przeglądarkom podjęcie decyzji, który program pomocniczy uruchomić). Szczegółowych informacji na ten temat należy szukać w dokumentacji serwera.

Aby wypróbować mechanizm Java Web Start, zainstaluj Tomcat dostępny na stronie *http://jakarta.apache.org/tomcat*. Jest to kontener na serwlety i strony JSP, ale serwuje też strony internetowe. Jest wstępnie skonfigurowany do serwowania poprawnego typu MIME dla plików JNLP. W dalszej części tej sekcji zakładamy, że używany jest serwer Tomcat.

Wypróbujemy mechanizm Java Web Start na kalkulatorze z rozdziału 12. Wykonaj następujące czynności:

1. Skompiluj program.

```
javac -classpath .:/path/to/javaws.jar webstart/*.java
```

2. Utwórz plik JAR za pomocą poniższego polecenia:

```
jar cvfe Calculator.jar webstart.Calculator webstart/*.class
```

3. Przygotuj plik rozruchowy *Calculator.jnlp* o następującej treści:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost:8080/calculator/"
href="Calculator.jnlp">
    <information>
        <title>Przykładowy kalkulator</title>
        <vendor>Cay S. Horstmann</vendor>
        <description>Kalkulator</description>
        <offline-allowed/>
    </information>
    <resources>
        <java version="1.6.0+"/>
        <jar href="Calculator.jar"/>
    </resources>
    <application-desc/>
</jnlp>
```

   (Należy pamiętać, że numer wersji to 1.6.0, a nie 6.0).

   Format pliku JNLP jest bardzo prosty. Jego pełna specyfikacja znajduje się na stronie *http://www.oracle.com/technetwork/java/javase/javawebstart*.

4. Utwórz katalog ***tomcat****/webapps/calculator*, z którego będzie serwowana aplikacja. W ścieżce tej *tomcat* to katalog główny instalacji Tomcata. Utwórz podkatalog ***tomcat****/webapps/calculator/WEB-INF* i umieść w nim poniższy plik *web.xml*:

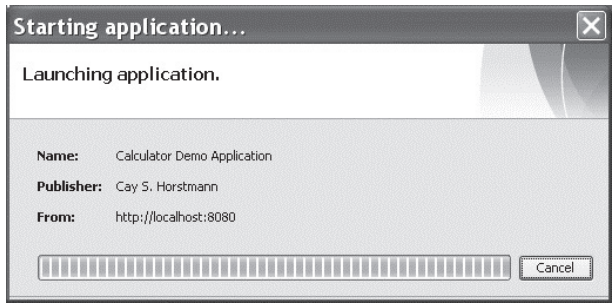
```
<?xml version="1.0" encoding="utf-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd">
</web-app>
```

5. Umieść pliki JAR i uruchomieniowy w katalogu ***tomcat****/webapps/calculator*.

6. Dodaj adres *http://localhost:8080* do listy zaufanych miejsc w panelu kontrolnym Javy, postępując zgodnie z instrukcjami zawartymi w podrozdziale 2.5, „Tworzenie i uruchamianie apletów”. Ewentualnie możesz zamiast tego podpisać plik JAR zgodnie z instrukcjami zamieszczonymi w sekcji 13.4.9, „Podpisywanie kodu”.

7. Uruchom serwer Tomcat, wykonując skrypt rozruchowy znajdujący się w katalogu ***tomcat****/bin*.

8. Wpisz w przeglądarce adres pliku JNLP. Na przykład w przypadku użycia serwera Tomcat należy wpisać adres *http://localhost:8080/calculator/Calculator.jnlp*. Jeżeli przeglądarka została skonfigurowana pod kątem pracy z Java Web Start, powinno się pojawić okno uruchamiania Java Web Start (rysunek 13.7).



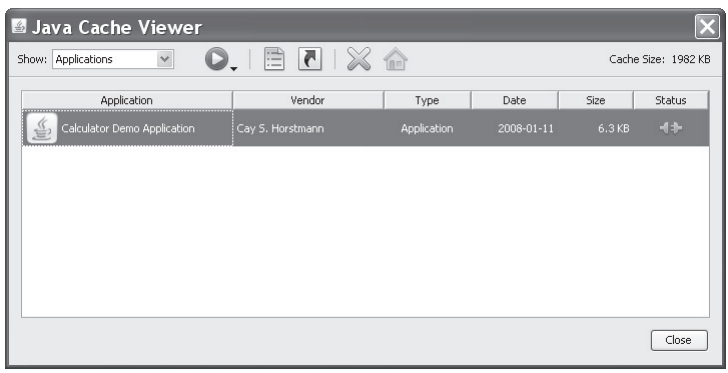
Rysunek 13.7. Uruchamianie aplikacji Java Web Start

Jeśli przeglądarka nie rozpoznaje plików JNLP, może zaproponować powiązanie ich z jakąś aplikacją. W takim przypadku wybierz ***jdk/bin/javaws***. W przeciwnym razie musisz dowiedzieć się, jak powiązać typ MIME `application/x-java-jnlp-file` z aplikacją `javaws`. Ewentualnie możesz jeszcze raz zainstalować pakiet JDK, który powinien to zrobić automatycznie.

9. Chwilę później powinien się pojawić kalkulator z informacją, że jest to aplikacja Javy (rysunek 13.8).

Rysunek 13.8. Kalkulator otwarty za pośrednictwem Java Web Start

10. Kiedy następnym razem spróbujesz uzyskać dostęp do pliku JNLP, program będzie pobierany z pamięci podręcznej. Zawartość tej pamięci można obejrzeć za pomocą panelu kontrolnego w postaci wtyczki Javy (rysunek 13.9). W systemie Windows wtyczki tej należy szukać w *Panelu sterowania*. W systemie Linux należy wykonać polecenie ***jdk****/jre/bin/ControlPanel*.



Rysunek 13.9. Aplikacje w pamięci podręcznej

Aby nie uruchamiać serwera podczas testowania konfiguracji JNLP, można tymczasowo nadpisać adres URL `codebase` w pliku JNLP za pomocą poniższego polecenia:

```
javaws -codebase file://katalogProgramu plikJNL
```

Na przykład w systemie Unix polecenie to można wydać w katalogu zawierającym plik JNLP:

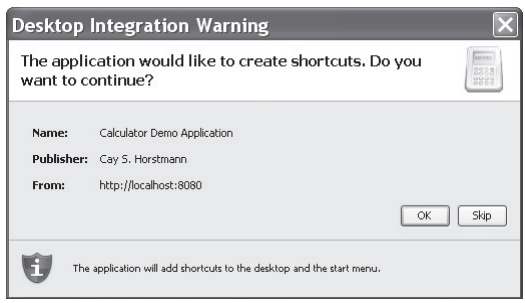
```
javaws -codebase file://`pwd` Calculator.jnlp
```

Oczywiście nie chcemy nakłaniać użytkowników do uruchamiania przeglądarki pamięci podręcznej za każdym razem, kiedy chcą uruchomić naszą aplikację. Można sprawić, aby instalator proponował utworzenie skrótów w menu *Start* i na pulpicie. W tym celu należy dodać poniższy kod do pliku JNLP:

```
<shortcut>
  <desktop/>
  <menu submenu="Akcesoria"/>
</shortcut>
```

Podczas pierwszego pobierania aplikacji zostanie wyświetlone ostrzeżenie o dodawaniu skrótów (rysunek 13.10).

Rysunek 13.10. Ostrzeżenie o integracji



Dodatkowo powinno się dodać ikonę dla skrótu i ekranu uruchomieniowego. Firma Sun zaleca stosowanie ikon o rozmiarach 32×32 i 64×64 piksele. Pliki ikon należy umieścić na serwerze razem z plikami JAR i JNLP. Poniższy kod należy dodać do sekcji `information` pliku JNLP:

```
<icon href="calc_icon32.png" width="32" height="32" />
<icon href="calc_icon64.png" width="64" height="64" />
```

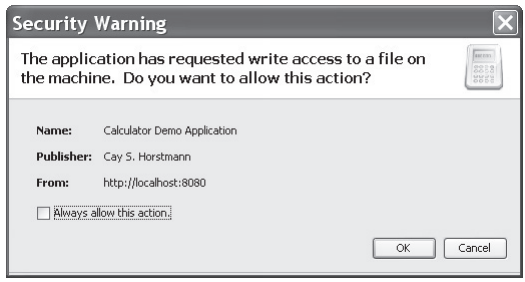
Należy pamiętać, że ikony te nie są związane z ikoną aplikacji. Aby wstawić ikonę dla aplikacji, należy dodać odrębny plik ikony do pliku JAR i wywołać metodę `IconImage` na rzecz klasy ramowej (zobacz listing 13.1).

13.5.2. API JNLP

Technologia Java Web Start ma tę przewagę nad apletami, że dla aplikacji działających w piaskownicy posiada API udostępniające przydatne usługi. Za jego pomocą aplikacje działające w piaskownicy mają dostęp do zasobów lokalnych przy zachowaniu odpowiedniego poziomu bezpieczeństwa. Istnieją na przykład usługi polegające na wysyłaniu i zapisywaniu plików.

Aplikacja nie ma dostępu do systemu plików i nie może określać nazw plików. W zamian wyświetlane jest okno dialogowe wyboru plików, w którym użytkownik programu wybiera plik. Przed wyświetleniem tego okna pojawia się ostrzeżenie i użytkownik musi wyrazić zgodę na kontynuowanie (rysunek 13.11). Ponadto opisywane API nie daje w rzeczywistości programowi dostępu do obiektu typu `File`. W szczególności aplikacja nie może sprawdzić lokalizacji pliku. W ten sposób programista zyskuje narzędzia do implementacji akcji otwierania i zapisywania plików, ale tak dużo informacji systemowych, jak to tylko możliwe, jest ukrytych przed niezaufanymi aplikacjami.

Rysunek 13.11. Ostrzeżenie Java Web Start



To API udostępnia następujące usługi:

- wysyłanie i zapisywanie plików,
- dostęp do schowka,
- drukowanie,
- pobieranie plików,
- wyświetlanie dokumentów w domyślnej przeglądarce,
- zapisywanie i odczytywanie stałych danych konfiguracyjnych,
- pilnowanie, aby był uruchomiony tylko jeden egzemplarz programu.

Dostęp do usługi uzyskuje się za pomocą metody `ServiceManager`:

```
FileSaveService service = (FileSaveService)
➥ServiceManager.lookup("javax.jnlp.FileSaveService");
```

Powyższa instrukcja spowoduje wyjątek `UnavailableServiceException`, jeśli usługa jest niedostępna.

Aby móc kompilować programy wykorzystujące API JNLP, należy umieścić plik *javaws.jar* na ścieżce klas. Plik ten znajduje się w podkatalogu *jre/lib* w katalogu JDK.

Omówimy najbardziej przydatne usługi JNLP. Aby zapisać plik, trzeba podać ścieżkę startową i rozszerzenia plików wyświetlanych w oknie dialogowym, dane do zapisania oraz sugerowaną nazwę pliku. Na przykład:

```
service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
```

Dane muszą być dostarczone w strumieniu `InputStream`, co nie zawsze jest łatwe. W programie na listingu 13.6 przyjęto następującą strategię działania:

1. Utworzenie strumienia `ByteArrayOutputStream` przechowującego bajty, które mają być zapisane.
2. Utworzenie strumienia `PrintStream` wysyłającego swoje dane do strumienia `ByteArrayOutputStream`.
3. Wydrukowanie informacji, które mają być zapisane, do strumienia `PrintStream`.
4. Utworzenie strumienia `ByteArrayInputStream` odczytującego zapisane bajty.
5. Przekazanie strumienia do metody `saveFileDialog`.

Więcej na temat strumieni piszemy w rozdziale 1. drugiego tomu. Teraz wystarczy tylko pobieżnie przejrzeć przykładowy program.

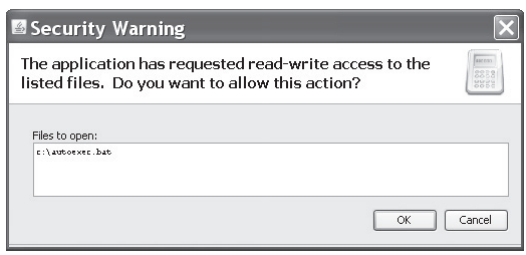
Do odczytu danych z pliku służy klasa `FileOpenService`. Jej metoda `openFileDialog` odbiera sugerowaną ścieżkę początkową i rozszerzenia plików wyświetlanych w oknie dialogowym i zwraca obiekt typu `FileContents`. Następnie można za pomocą metod `getInputStream` i `getOutputStream` odczytać i zapisać dane do pliku. Jeśli użytkownik nie wybrał pliku, metoda `openFileDialog` zwraca wartość `null`.

```
FileOpenService service = (FileOpenService)
ServiceManager.lookup("javax.jnlp.FileOpenService");
FileContents contents = service.openFileDialog(".", new String[] { "txt" });
if (contents != null)
{
  InputStream in = contents.getInputStream();
  . . .
}
```

Pamiętajmy, że aplikacja nie zna nazwy ani lokalizacji pliku. Aby otworzyć określony plik, można użyć klasy `ExtendedService`:

```
ExtendedService service = (ExtendedService)
ServiceManager.lookup("javax.jnlp.ExtendedService");
FileContents contents = service.openFile(new File("c:\\autoexec.bat"));
if (contents != null)
{
  OutputStream out = contents.getOutputStream();
  . . .
}
```

Użytkownik programu musi się zgodzić na dostęp do tego pliku (rysunek 13.12).



Rysunek 13.12. Ostrzeżenie o dostępie do pliku

Do wyświetlenia dokumentu w domyślnej przeglądarce należy użyć interfejsu `BasicService`. Zauważmy, że niektóre systemy mogą nie mieć domyślnej przeglądarki.

```
BasicService service = (BasicService) ServiceManager.lookup("javax.jnlp.BasicService");
if (service.isWebBrowserSupported())
  service.showDocument(url);
else . . .
```

Podstawowy interfejs `PersistenceService` pozwala aplikacji zapisywać niewielkie ilości danych konfiguracyjnych i odczytywać je po jej ponownym uruchomieniu. Mechanizm ten przypomina nieco pliki cookie HTTP. Ten stały magazyn jako klucze wykorzystuje adresy URL. Adres URL nie musi prowadzić do istniejącego zasobu sieciowego. Są one w tym przypadku wykorzystywane jako wygodny sposób nazewnictwa hierarchicznego. Dla każdego klucza URL aplikacja może zapisać dowolne dane binarne (rozmiar jednego bloku danych w magazynie może być ograniczony).

Odseparowanie od siebie poszczególnych aplikacji jest możliwe dzięki temu, że każdy program może używać tylko takich kluczy, które zaczynają się od określonego podstawowego łańcucha (zgodnie z informacjami w pliku JNLP). Jeśli na przykład aplikacja zostanie pobrana ze strony *http://myserver.com/apps*, może używać tylko kluczy w formacie *http://myserver.com/apps/subkey1/subkey2/…*. Próby dostępu do innych kluczy zakończą się niepowodzeniem.

Aplikacja może sprawdzić podstawę swojego klucza URL za pomocą metody `getCodeBase` z klasy `BasicService`.

Do tworzenia kluczy służy metoda `create` z interfejsu `PersistenceService`.

```
URL url = new URL(codeBase, "mykey");
service.create(url, maxSize);
```

Aby uzyskać informacje związane z określonym kluczem, należy wywołać metodę `get`. Zwraca ona obiekt typu `FileContents`, za pośrednictwem którego można odczytywać i zapisywać dane kluczy. Na przykład:

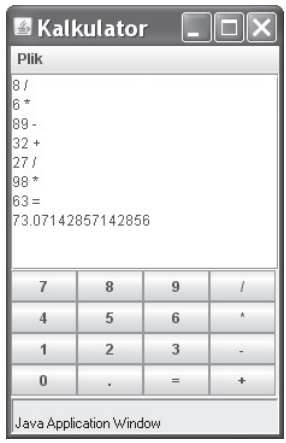
```
FileContents contents = service.get(url);
InputStream in = contents.getInputStream();
OutputStream out = contents.getOutputStream(true);    // true = nadpisz
```

Niestety nie ma dobrego sposobu na sprawdzenie, czy określony klucz już istnieje, czy trzeba go utworzyć. Można wywołać metodę `get`. Jeśli klucz nie istnieje, zostanie spowodowany wyjątek `FileNotFoundException`.

Aplikacje Java Web Start i aplety mogą drukować, wykorzystując normalne API drukowania. Pojawia się tylko okienko, w którym użytkownik jest proszony o zezwolenie na dostęp do drukarki. Więcej informacji na temat API drukowania znajduje się w rozdziale 7. drugiego tomu.

Program na listingu 13.6 jest prostym rozszerzeniem wcześniej utworzonego kalkulatora. Ma on wirtualną taśmę, która rejestruje wszystkie obliczenia. Można zapisywać i ładować historię obliczeń. Aplikacja pozwala na ustawienie tytułu ramki, co służy demonstracji trwałego magazynu. Przy ponownym uruchomieniu aplikacja pobierze wybrany przez użytkownika tytuł z trwałego magazynu (rysunek 13.13).

Rysunek 13.13. Aplikacja WebStartCalculator



Listing 13.6. webstart/CalculatorFrame.java

```
package webstart;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.MalformedURLException;
import java.net.URL;

import javax.jnlp.BasicService;
import javax.jnlp.FileContents;
import javax.jnlp.FileOpenService;
import javax.jnlp.FileSaveService;
import javax.jnlp.PersistenceService;
import javax.jnlp.ServiceManager;
import javax.jnlp.UnavailableServiceException;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;


/**
 * Ramka z kalkulatorem i menu do ładowania oraz zapisywania historii obliczeń
 */
public class CalculatorFrame extends JFrame
{
   private CalculatorPanel panel;

   public CalculatorFrame()
   {
      setTitle();
```

```
      panel = new CalculatorPanel();
      add(panel);

      JMenu fileMenu = new JMenu("Plik");
      JMenuBar menuBar = new JMenuBar();
      menuBar.add(fileMenu);
      setJMenuBar(menuBar);

      JMenuItem openItem = fileMenu.add("Otwórz");
      openItem.addActionListener(event -> open());
      JMenuItem saveItem = fileMenu.add("Zapisz");
      saveItem.addActionListener(event -> save());

      pack();
   }

   /**
   * Pobiera tytuł z magazynu trwałego lub prosi użytkownika o podanie tytułu, jeśli
   * nie ma wcześniejszego wpisu.
   */
   public void setTitle()
   {
      try
      {
         String title = null;

         BasicService basic = (BasicService)
         ➥ServiceManager.lookup("javax.jnlp.BasicService");
         URL codeBase = basic.getCodeBase();

         PersistenceService service = (PersistenceService) ServiceManager
               .lookup("javax.jnlp.PersistenceService");
         URL key = new URL(codeBase, "title");

         try
         {
            FileContents contents = service.get(key);
            InputStream in = contents.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            title = reader.readLine();
         }
         catch (FileNotFoundException e)
         {
            title = JOptionPane.showInputDialog("Podaj tytuł ramki:");
            if (title == null) return;

            service.create(key, 100);
            FileContents contents = service.get(key);
            OutputStream out = contents.getOutputStream(true);
            PrintStream printOut = new PrintStream(out);
            printOut.print(title);
         }
         setTitle(title);
      }
      catch (UnavailableServiceException e)
      {
         JOptionPane.showMessageDialog(this, e);
      }
```

```
   }

   /**
   * Otwiera plik historii i aktualizuje zawartość wyświetlacza.
   */
   public void open()
   {
      try
      {
         FileOpenService service = (FileOpenService) ServiceManager
               .lookup("javax.jnlp.FileOpenService");
         FileContents contents = service.openFileDialog(".", new String[] { "txt" });

         JOptionPane.showMessageDialog(this, contents.getName());
         if (contents != null)
         {
            InputStream in = contents.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            String line;
            while ((line = reader.readLine()) != null)
            {
               panel.append(line);
               panel.append("\n");
            }
         }
      }
      catch (UnavailableServiceException e)
      {
         JOptionPane.showMessageDialog(this, e);
      }
      catch (IOException e)
      {
         JOptionPane.showMessageDialog(this, e);
      }
   }

   /**
   * Zapisuje historię kalkulatora w pliku.
   */
   public void save()
   {
      try
      {
         ByteArrayOutputStream out = new ByteArrayOutputStream();
         PrintStream printOut = new PrintStream(out);
         printOut.print(panel.getText());
         InputStream data = new ByteArrayInputStream(out.toByteArray());
         FileSaveService service = (FileSaveService) ServiceManager
               .lookup("javax.jnlp.FileSaveService");
         service.saveFileDialog(".", new String[] { "txt" }, data, "calc.txt");
      }
      catch (UnavailableServiceException e)
      {
         JOptionPane.showMessageDialog(this, e);
      }
      catch (IOException e)
      {
```

```
            JOptionPane.showMessageDialog(this, e);
         }
      }
   }
```

`javax.jnlp.ServiceManager`

- `static String[] getServiceNames()`

  Zwraca nazwy wszystkich dostępnych usług.

- `static Object lookup(string name)`

  Zwraca usługę o podanej nazwie.

`javax.jnlp.BasicService`

- `URL getCodeBase()`

  Zwraca katalog zawierający kod aplikacji.

- `boolean isWebBrowserSupported()`

  Zwraca wartość `true`, jeśli środowisko Web Start może uruchomić przeglądarkę.

- `boolean showDocument(URL url)`

  Podejmuje próbę pokazania danego adresu URL w przeglądarce. Zwraca wartość `true`, jeśli żądanie kończy się powodzeniem.

`javax.jnlp.FileContents`

- `InputStream getInputStream()`

  Zwraca strumień wejściowy do odczytu zawartości pliku.

- `OutputStream getOutputStream(boolean overwrite)`

  Zwraca strumień wyjściowy do zapisu do pliku. Jeśli parametr `overwrite` ma wartość `true`, aktualna treść pliku jest nadpisywana.

- `String getName()`

  Zwraca nazwę pliku (nie pełną ścieżkę katalogową).

- `boolean canRead()`

- `boolean canWrite()`

  Zwraca wartość `true`, jeśli dany plik nadaje się do odczytu lub zapisu.

`javax.jnlp.FileOpenService`

- `FileContents openFileDialog(String pathHint, String[] extensions)`

- `FileContents[] openMultiFileDialog(String pathHint, String[] extensions)`

  Wyświetla ostrzeżenie dla użytkownika i okno wyboru pliku. Zwraca deskryptory treści pliku lub plików wybranych przez użytkownika bądź wartość `null`, jeśli użytkownik nie wybrał żadnego pliku.

`javax.jnlp.FileSaveService`

- `FileContents saveFileDialog(String pathHint, String[] extensions, InputStream data,String nameHint)`
- `FileContents saveAsFileDialog(String pathHint, String[] extensions, FileContents data)`

  Wyświetla ostrzeżenie dla użytkownika i okno wyboru pliku. Zapisuje dane i zwraca deskryptory treści pliku lub plików wybranych przez użytkownika bądź wartość `null`, jeśli użytkownik nie wybrał żadnego pliku.

`javax.jnlp.PersistenceService`

- `long create(URL key, long maxsize)`

  Zapisuje dany klucz w pamięci trwałej. Zwraca maksymalny rozmiar przyznawany przez pamięć trwałą.

- `void delete(URL key)`

  Usuwa dany klucz.

- `String[] getNames(URL url)`

  Zwraca względne nazwy wszystkich kluczy, które zaczynają się od danego adresu URL.

- `FileContents get(URL key)`

  Tworzy deskryptor treści, za pośrednictwem którego można modyfikować dane związane z danym kluczem. Jeśli dla danego klucza nie istnieje żaden wpis, zgłaszany jest wyjątek `FileNotFoundException`.

Na tym kończymy opis technik przygotowywania aplikacji w Javie do użytku. W ostatnim rozdziale zajmiemy się bardzo ważnym tematem współbieżności.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Z

Ź

POZNAJ JAVĘ — IDEALNY JĘZYK DLA NAJLEPSZYCH PROGRAMISTÓW!

Niemal od początku swojego istnienia Java była traktowana jako wszechstronne narzędzie do budowy bezpiecznych aplikacji internetowych. Zaaprobowała ją znacząca większość poważnych firm informatycznych. Dziś jest uważana za niezwykle starannie zaprojektowany i rozwijany język, w który wbudowano wyrafinowane funkcje ułatwiające implementację wielu złożonych zadań programistycznych. Aby w pełni wykorzystać możliwości Javy, trzeba zrozumieć rządzące nią zasady i poznać jej zaawansowane cechy.

Książka, którą trzymasz w dłoni, to poważny podręcznik dla poważnych programistów. Opisano tu podstawy języka oraz najważniejsze zagadnienia związane z programowaniem interfejsu użytkownika. Przedstawiono pakiet Java Development Kit. Pakiet ten obejmuje obecnie tak różne aspekty tworzenia aplikacji, jak konstruowanie interfejsu użytkownika, zarządzanie bazami danych, internacjonalizacja, bezpieczeństwo i przetwarzanie XML. W książce znajdziesz również mnóstwo przykładów kodu obrazujących zasady działania niemal każdej opisywanej funkcji i biblioteki.

NAJWAŻNIEJSZE ZAGADNIENIA:

- programowanie obiektowe
- mechanizm refleksji i obiekty proxy
- interfejsy i klasy wewnętrzne
- system kolekcji
- projektowanie graficznego interfejsu użytkownika za pomocą pakietu narzędzi Swing UI
- współbieżność

CAY S. HORSTMANN — jest autorem wielu książek dotyczących programowania w Javie, przeznaczonych dla zawodowych programistów oraz studentów informatyki. Jest profesorem informatyki na Uniwersytecie Stanowym w San Jose. Zdobył tytuł Java Champion. Często wygłasza referaty na konferencjach informatycznych.

Helion
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

PRENTICE HALL
PEARSON EDUCATION

ISBN 978-83-283-2480-0
9 788328 324800
cena: 99,00 zł

sięgnij po WIĘCEJ
KOD KORZYŚCI

ALWAYS LEARNING
PEARSON