

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Po prostu Java 2

Autor: Dori Smith

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 83-7197-686-0

Tytuł oryginału: [Java 2 for the World Wide Web VQG](#)

Format: B5, stron: 356



Java jest jednym z najpopularniejszych języków programowania na świecie wykorzystywanym przy tworzeniu popularnych aplikacji. Język ten dla początkujących programistów jest często pierwszym poznany językiem programowania. Dzięki przemyślanej architekturze obiektowej Javy łatwiej Ci będzie nabrać prawidłowych nawyków programistycznych, z kolei funkcjonalność tego języka sprawi, że nauka nie pójdzie na marne, a zdobytą wiedzę wykorzystasz w praktyce. W Javie można bowiem napisać prawie każdą aplikację, od prostej gry działającej w telefonie komórkowym, po skomplikowany system uruchamiany na potężnym serwerze.

„Po prostu Java 2” to książka, dzięki której nauczysz się pisać programy w tym języku, nawet jeśli programowanie było Ci do tej pory zupełnie obce. Książka jest pozbawiona zbędnego balastu teorii. Prezentuje ona wiele zastosowań Javy.

Dzięki książce nauczysz się:

- Składni Javy oraz podstawowych instrukcji
- Korzystać z obiektów i rozpoznawać ich metody
- Tworzyć interfejs użytkownika dla aplikacji Javy
- Pisać aplety i osadzać je w stronach WWW
- Łączyć Javę z JavaScript
- Pisać servlety i strony JSP
- Korzystać z XML z poziomu Javy

Chcesz szybko i bez zbędnych dywagacji dowiedzieć się, dlaczego Java cieszy się takim powodzeniem? Chcesz wykorzystać ją do swoich potrzeb? Ta książka z pewnością Ci w tym pomoże.



Spis treści

Wstęp	11
Rozdział 1. Czym jest Java?	17
Krótka lekcja historii.....	18
Dlaczego uczyć się Javy?.....	20
Rozpowszechnione mity o Javie.....	21
Co będzie Ci potrzebne.....	24
Wersje Javy.....	25
Rozdział 2. Podstawy Javy	27
Wprowadzenie w niesamowity świat Wirtualnej Maszyny Javy.....	28
Bezpieczeństwo Javy	29
Pliki Javy.....	30
Gra w Buzzword Bingo	31
Rozdział 3. Używanie Javy na stronach WWW	37
Znacznik APPLET	38
Znacznik OBJECT	40
Znacznik OBJECT połączony ze znacznikiem EMBED	43
Wyświetlanie komunikatów w przeglądarkach nie obsługujących Javy ..	45
Ustawianie położenia apletów Javy na stronach WWW.....	47
Znajdowanie apletów w Internecie	49
Przekazywanie parametrów do apletu.....	51
Rozdział 4. Twoje pierwsze aplety	55
„Witaj świecie!”	56
Posługiwanie się czcionkami	58
Ustawianie koloru tła	59
Przekazywanie parametrów z pliku HTML do apletu Javy	60
Dodawanie komentarzy	62
Zmiana stylu czcionki	63
Zmiana kolorów	64

Rozdział 5. Łańcuchy	65
Zmienne łańcuchowe	66
Więcej o łańcuchach	68
Zasięg deklaracji zmiennych.....	70
Metody klasy String.....	72
Wszystkie typy liczbowe	74
Wzajemne przekształcanie łańcuchów i liczb.....	77
Przekształcenia typów liczbowych	79
Przekształcanie przez rzutowanie	81
Tablice obiektów.....	83
Rozdział 6. Podejmowanie decyzji	85
Instrukcje warunkowe — if	86
Przechwytywanie błędów	89
Więcej o instrukcjach warunkowych — if...else	93
Powtarzanie czynności za pomocą pętli.....	96
Inny rodzaj pętli — while	98
Ostatni rodzaj pętli — do...while	100
Przerywanie działania pętli	102
Konstrukcja switch...case.....	104
Rozdział 7. Współpraca z użytkownikiem	109
Rysowanie za pomocą myszy	110
Rysowanie w dwóch wymiarach	113
Swobodne rysowanie	116
Przechwytywanie naciskania klawiszy	118
Przemieszczanie obrazów	120
Używanie klawiszy modyfikujących	122
Rozdział 8. Budowa interfejsu użytkownika	125
Strony chronione hasłem.....	126
Wpisywanie i wyświetlanie tekstu.....	129
Praca z polami wyboru.....	131
Używanie przycisków opcji.....	134
Elementy rozwijanego menu.....	137
Ustawianie menu czcionki w Javie	139
Używanie wieloliniowych pól tekstowych	141
Używanie list przewijanych.....	144

Rozdział 9. Rozmieszczanie elementów interfejsu użytkownika	147
Brak rozmieszczenia — FlowLayout.....	148
Używanie menadżera FlowLayout.....	150
Wstawianie wolnych przestrzeni w obszarze apletu.....	152
Używanie menadżera BorderLayout.....	154
Używanie menadżera GridLayout	156
Używanie wkładek.....	158
Dodawanie komponentów z wykorzystaniem paneli.....	160
Używanie menadżera CardLayout	162
Używanie menadżera GridBagLayout	165
Rozdział 10. Manipulowanie obrazami i animacje	169
Wyświetlanie obrazu.....	170
Wyświetlanie fragmentu obrazu	172
Rysowanie ramki wokół apletu.....	174
Wyświetlanie wielu obrazów	176
Wątki i animacja	179
Podwójnie buforowane animacje	183
Wyświetlanie nieskończonej liczby obrazów	186
Kontrola animacji.....	188
Rozdział 11. Java i JavaScript	191
Sprawdzanie obsługi Javy przez przeglądarkę.....	192
JavaScript i publiczne metody Javy	194
Przekazywanie danych z JavaScriptu do Javy	197
Przekazywanie danych z Javy do JavaScriptu	200
Rozdział 12. Projektowanie interfejsu użytkownika z komponentami Swing	203
Twój pierwszy aplet wykorzystujący komponent Swing.....	204
Ustawianie czcionek dla komponentów Swing	206
Pola wyboru Swing	208
Przyciski opcji Swing	211
Wygląd apletu zgodny z preferencjami użytkownika.....	214
Animacja i Swing.....	219
Swing w akcji.....	223

Rozdział 13. JavaServer Pages i Serwlety Javy	231
Twoja pierwsza strona JSP	232
JSP i formularze.....	234
Instrukcje warunkowe w JSP	236
Zapisywanie cookies za pomocą JSP	238
Odczytywanie cookies za pomocą JSP	240
XML i JSP	243
Twój pierwszy serwlet	245
Ankieta i serwlety	248
Rozdział 14. Java i narzędzia wizualne	257
Wstawianie znacznika applet w programie Dreamweaver	258
Wstawianie znacznika object w programie Dreamweaver	261
Wstawianie znacznika applet w programie GoLive	263
Wstawianie znacznika object w programie GoLive	265
Rozdział 15. Kółko i krzyżyk	267
Gra w kółko i krzyżyk.....	268
Rozdział 16. Prosty kalkulator	283
Aplet z prostym kalkulatorem.....	284
Rozdział 17. Prawdziwy świat Javy — hierarchiczne menu	291
Java i hierarchiczne menu	292
Dodatek A Gdzie szukać dodatkowych informacji?	307
Java w Internecie.....	308
Czasopisma internetowe	311
Zintegrowane środowiska oprogramowania	312
Książki o Javie	314
Grupy dyskusyjne	316
Witryny internetowe a przenośność Javy.....	317
Dodatek B Zarezerwowane słowa kluczowe	319
Zarezerwowane słowa kluczowe Javy	320
Dodatek C Hierarchia obiektów Javy	323
Pakiet java.applet	324
Pakiet java.awt	324
Pakiet java.awt.color (wprowadzony w JDK 1.2)	325
Pakiet java.awt.datatransfer (wprowadzony w JDK 1.1).....	326

Pakiet java.awt.dnd (wprowadzony w JDK 1.2).....	326
Pakiet java.awt.event (wprowadzony w JDK 1.1)	327
Pakiet java.awt.font (wprowadzony w JDK 1.2)	327
Pakiet java.awt.geom (wprowadzony w JDK 1.2).....	328
Pakiet java.awt.im (wprowadzony w JDK 1.2)	328
Pakiet java.awt.image (wprowadzony w JDK 1.2).....	329
Pakiet java.awt.image.renderable (wprowadzony w JDK 1.2)	330
Pakiet java.awt.peer (od wersji JDK 1.1, nie powinno się bezpośrednio używać interfejsów tego pakietu)	330
Pakiet java.awt.print (wprowadzony w JDK 1.2)	331
Pakiet java.beans (wprowadzony w JDK 1.1)	331
Pakiet java.beans.beancontext (wprowadzony w JDK 1.2)	332
Pakiet java.io.....	333
Pakiet java.lang	334
Pakiet java.lang.ref (wprowadzony w JDK 1.2)	335
Pakiet java.lang.reflect (wprowadzony w JDK 1.1)	335
Pakiet java.math (wprowadzony w JDK 1.1)	336
Pakiet java.net.....	336
Pakiet java.text (wprowadzony w JDK 1.1)	336
Pakiet java.util	337
Pakiet java.util.jar (wprowadzony w JDK 1.2).....	337
Pakiet java.util.zip (wprowadzony w JDK 1.1)	338
Dodatek D Różnice pomiędzy JDK 1.0, 1.1, 1.2 i 1.3	339
„Stary” model zdarzeń	340
Skorowidz	347

Podejmowanie decyzji

6

Nie zawsze aplety Javy przechodzą najprostszą drogę od pierwszej do ostatniej instrukcji. Niekiedy potrzebne jest ominięcie fragmentu kodu, czy też powtórne uruchomienie pewnych instrukcji. Wpływanie na przebieg działania programu nazywa się *sterowaniem przepływem* (ang. *flow control*). W tym rozdziale omówimy, jak zapanować nad kierunkiem przetwarzania Twojego kodu.

Instrukcje warunkowe — if

Najprostszym sposobem przejęcie kontroli nad działaniem apletu jest użycie instrukcji `if`. Działanie tej instrukcji można wyrazić słowami: „Jeśli (`if`) jakaś wartość jest prawdziwa (`true`), wówczas uruchom instrukcje znajdujące się w nawiasach klamrowych. W przeciwnym wypadku pomini ten fragment kodu”. Listing HTML 6.1 i aplet 6.1 pokazują, jak używać instrukcji warunkowych w celu wyświetlenia wpisanego przez użytkownika tekstu.

Aby skorzystać z instrukcji `if`, należy wpisać:

1. `import java.awt.event.*;`

Na początku musimy obsłużyć zdarzenia generowane przez użytkownika. Konieczne do tego będą klasy z pakietu `java.awt.event`, które w tym kroku importujemy.

2. `public class Applet1 extends Applet`
`↳ implements ActionListener {`

Nasza nowa klasa (nazwana `Applet1`) została zdefiniowana jako rozszerzenia standardowej klasy `Applet`. Nowością w tej definicji jest słowo kluczowe `implements ActionListener`. Oznacza to, że nasz aplet będzie zdolny do obsługi zdarzeń generowanych przez użytkownika i będzie przez cały czas swojego działania oczekiwał na jego konkretne czynności. Tabela 6.1 przedstawia metody służące do przechwytywania interakcji z użytkownikiem dla poszczególnych elementów.

3. `TextField inField = new TextField(12);`
`Font f = new Font("TimesRoman", Font.`
`↳ BOLD, 24)`
`String wpisanyTekst = "";`

Oto, jak tworzymy nowe zmienne: pole tekstowe `inField`, przeznaczone do wpisywania znaków przez użytkownika, zmienną czcionki `f`, którą ustawiamy na rozmiar 24-punktowy i pogrubiony krój Times Roman, oraz łańcuch `wpisanyTekst`, który będzie zawierał tekst wpisany przez użytkownika i wypisywany później na ekranie (na początku ustawiamy łańcuch jako pusty).

Listing HTML 6.1. Umieszcza aplet 6.1 w oknie przeglądarki

```
Listing
<html>
<head>
  <title>Listing 6.1</title>
</head>
<body bgcolor="white">
<object classid="clsid:8AD9C840-044E-11D1-
↳ B3E9-00805F499D93" width="500" height="100"
↳ codetype="application/java">
  <param name="code" value="Applet1.class">
  <param name="type" value="application/
↳ x-java-applet;version=1.3">
  <param name="scriptable" value="false">
  <embed type="application/x-java-applet;
↳ version=1.3" code="Applet1.class" width=
↳ "500" height="100" scriptable="false"
↳ pluginspage="http://java.sun.com/products/
↳ plugin/1.3/plugin-install.html">
  </embed>
</object>
</body>
</html>
```

Aplet 6.1. Jeśli wpiszesz tekst do pola `inField`, Java wyświetli go na ekranie

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

public class Applet1 extends Applet implements
↳ ActionListener {
  TextField inField = new TextField(12);
  Font f = new Font("TimesRoman", Font.BOLD, 24);
  String wpisanyTekst = "";

  public void init() {
    setBackground(Color.white);
    inField.addActionListener(this);
    add(inField);
  }

  public void paint(Graphics g) {
    g.setFont(f);
    if (wpisanyTekst != "") {
      g.drawString("Wpisane: " + wpisanyTekst,
↳ 20, 60);
    }
  }

  public void actionPerformed(ActionEvent e) {
    wpisanyTekst = inField.getText();
    repaint();
  }
}
```




Rysunek 6.1. Oto aplet wyświetlający wpisany przez użytkownika tekst

Wskazówki

- Jeśli programowałeś wcześniej w innych językach, możesz sądzić, że konieczne jest użycie instrukcji `then` po instrukcji `if`. W Javie nie używa się tej instrukcji.
- Bardzo często w programach Javy można spotkać instrukcję `if` poprzedzającą pojedynczą, wciętą instrukcję nie umieszczoną w nawiasach klamrowych. Taka poprawna składnia oznacza, że instrukcja warunkowa `if` dotyczy tylko jednej, następującej po niej instrukcji. Dla zachowania przejrzystości kodu, w książce tej zawsze będziemy następujące po `if` instrukcje umieszczać w nawiasach klamrowych.

4. `inField.addActionListener(this);`
`add(inField);`

Teraz wstawiamy dwie nowe instrukcje metodzie `init()`. Pierwsza określa, że chcemy użyć interfejsu `ActionListener` (omówionego w kroku 2.), by przechwytywać działania użytkownika w polu tekstowym `inField`. Dzięki temu, za każdym razem, kiedy użytkownika cokolwiek wpisze w tym polu, wyzwolone będzie odpowiednie zdarzenie w Javie (ang. *event*).

Druga linia spowoduje wyświetlenie pola tekstowego w obszarze apletu.

5. `if (wpisanyTekst != "") {`
 `g.drawString("Wpisales: " +`
 `wpisanyTekst, 20, 60);`
`}`

Wtedy, i tylko wtedy, gdy łańcuch `wpisanyTekst` zawiera jakikolwiek znak (długość łańcucha jest większa od zera), wyświetlamy tekst wpisany przez użytkownika. Jeśli łańcuch jest pusty, instrukcja `g.drawString()` nie zostanie wykonana.

6. `public void actionPerformed`
 `{(ActionEvent e) {`
 `wpisanyTekst = inField.getText();`
 `repaint();`
 `}`

Ponieważ dodaliśmy wcześniej `ActionListener` do pola tekstowego `inField`, metoda `actionPerformed()` będzie obsługiwać wszystkie zdarzenia wywołane przez użytkownika. Jedynym możliwym zdarzeniem w tym aplecie jest wpisanie tekstu w jedynym polu tekstowym. Kiedy to się stanie, przypisujemy zmiennej łańcuchowej `wpisanyTekst` dane z tego pola. Następnie wywołujemy metodę `repaint()`, która odświeża obszar apletu (efekt możemy zobaczyć na rysunku 6.1).

Rozdział 6.

Tabela 6.1. Zdarzenia w Javie

Interfejs	Komponenty	Metody
ActionListener	Button (przycisk)	ActionPerformed()
	List (lista)	
	TextField (pole tekstowe)	
	MenuItem (element menu)	
AdjustmentListener	Scrollbar (pasek przewijania)	adjustmentValueChanged()
ComponentListener	Wszystkie komponenty	componentHidden()
		ComponentMoved()
		componentResized()
		componentShown()
ContainerListener	Wszystkie pojemniki	componentAdded()
		componentRemoved()
FocusListener	Wszystkie komponenty	focusGained()
		focusLost()
ItemListener	Checkbox (pole wyboru)	itemStateChanged()
	CheckBoxMenuItem (pole wyboru elementu menu)	
	Choice (lista rozwijana)	
	ItemSelectable (element możliwy do wybrania)	
KeyListener	Wszystkie komponenty	List (lista)
		keyPressed()
		keyReleased()
		keyTyped()
MouseListener	Wszystkie komponenty	mouseClicked()
		mouseEntered()
		mouseExited()
		mousePressed()
		mouseReleased()
		mouseDragged()
MouseMotionListener	Wszystkie komponenty	mouseMoved()
		textValueChanged()
TextListener	TextComponent (komponent tekstowy)	windowActivated()
WindowListener	Window (okno)	windowClosed()
		windowClosing()
		windowDeactivated()
		windowDeiconified()
		windowIconified()
		windowOpened()

Listing HTML 6.2. *Wywołuje aplet 6.2 i umieszcza go w oknie przeglądarki internetowej*

```
Listing
<html>
<head>
  <title>Listing 6.2</title>
</head>
<body bgcolor="white">
  <object classid="clsid:8AD9C840-044E-11D1-
  ➤B3E9-00805F499D93" width="500" height="100"
  ➤codetype="application/java">
    <param name="code" value="Applet2.class">
    <param name="type" value="application/
    ➤x-java-applet;version=1.3">
    <param name="scriptable" value="false">
    <embed type="application/x-java-applet;
    ➤version=1.3" code="Applet2.class" width=
    ➤"500" height="100" scriptable="false"
    ➤pluginspage="http://java.sun.com/products/
    ➤plugin/1.3/plugin-install.html">
    </embed>
  </object>
</body>
</html>
```

Przechwytywanie błędów

Kiedy umożliwisz użytkownikowi interakcję z Twoim apletem, może się zdarzyć, że dane, które poda, nie będą pasowały do założeń Twojego programu. Listing HTML 6.2 i aplet 6.2 pokazują, jak przechwycić potencjalne błędy, zanim spowodują poważne problemy w działaniu apletu.

Aby przechwycić błędy, należy użyć:

1. `TextField` `poleZgadywania` = new
➤ `TextField(5);`
`int` `nastepnaProba` = -1;

Powyżej definiujemy zmienną `poleZgadywania` typu `TextField` (pole tekstowe o długości 5). Wykorzystamy to pole do pobierania od użytkownika zgadywanej liczby. Zmienna `nastepnaProba` posłuży nam do przechowywania wpisywanej przez użytkownika liczby — przypisujemy jej wartość początkową -1. Wartość jest na pewno niepoprawna (szukana liczba należy do przedziału od 1 do 100), zatem używamy -1, by program „wiedział”, że użytkownik nie podał jeszcze żadnej wartości.

2. `int` `szukanaLiczba` = (int)(`java.lang.`
➤ `Math.random() * 100`) + 1;

Metoda Javy `java.lang.Math.random()` generuje rzeczywistą liczbę losową z przedziału od 0 do 1 (np. 0,722 lub 0,111). Mnożymy tę wartość przez 100 (mamy więc 72,2 lub 11,1). Na końcu przekształcamy tę wartość na liczbę całkowitą (rzutujemy na typ `int`), co powoduje obcięcie części dziesiętnej naszej liczby (otrzymujemy zatem liczbę od 0 do 99). Po dodaniu 1 mamy wynik od 1 do 100 i przypisujemy taką właśnie wartość zmiennej `szukanaLiczba`. Zarówno zmienna `nastepnaProba`, jak i `szukanaLiczba` są typu `int`.

Rozdział 6.

3. `String statusLiczby = nastepnaProba + "`
`➡ jest szukaną liczbą";`
`g.setFont(f);`

Wewnątrz metody `paint()` zdefiniowaliśmy nową zmienną łańcuchową: `statusLiczby`. Przypisujemy jej wartość zmiennej `nastepnaProba` z dołączonym komunikatem o udanej próbie odgadnięcia liczby (patrz rysunek 6.2). W drugiej instrukcji ustawiamy czcionkę dla naszego apletu (za pomocą zmiennej `f`).

4. `if (nastepnaProba != szukanaLiczba) {`
`statusLiczby = nastepnaProba + " nie`
`➡ jest szukaną liczbą";`
`}`

Oto nasza pierwsza instrukcja `if`. Sprawdzamy w niej, czy liczba będąca wartością zmiennej `nastepnaProba` różni się od wylosowanej wcześniej wartości `szukanaLiczba`; sprawdzamy zatem, czy podana przez użytkownika liczba różni się od wartości, którą próbuje odgadnąć. Jeśli wynikiem porównania będzie `true` (użytkownikowi nie udało się odgadnąć naszej liczby), zostanie uruchomiony kod znajdujący się wewnątrz nawiasów klamrowych. W przeciwnym wypadku, ten fragment programu zostanie pominięty. Jeśli użytkownik nie odgadł wylosowanej liczby, ponownie ustawiamy zmienną `statusLiczby`, tym razem z komunikatem o nieudanej próbie (patrz rysunek 6.3).

Aplet 6.2. Zagrajmy w grę „zgadnij, jaką sobie wymyśliłem liczbę”

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet2 extends Applet implements
➡ ActionListener {
    TextField poleZgadywania = new TextField(5);
    int nastepnaProba = -1;
    int szukanaLiczba = (int)(java.lang.Math.
➡ random() * 100) + 1;
    Font f = new Font("TimesRoman", Font.BOLD,
➡ 24);

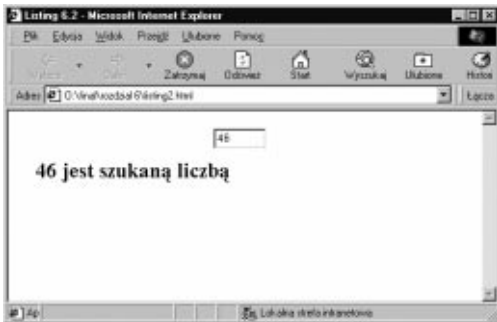
    public void init() {
        setBackground(Color.white);
        poleZgadywania.addActionListener(this);
        add(poleZgadywania);
    }

    public void paint(Graphics g) {
        String statusLiczby = nastepnaProba + "
➡ jest szukaną liczbą";

        g.setFont(f);
        if (nastepnaProba != szukanaLiczba) {
            statusLiczby = nastepnaProba + " nie
➡ jest szukaną liczbą";
        }
        if (nastepnaProba < 1) {
            statusLiczby = "Odgadnij liczbę od 1
➡ do 100";
        }
        g.drawString(statusLiczby, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                nastepnaProba = Integer.parseInt
➡ (poleZgadywania.getText());
            }
            catch (NumberFormatException x) {
                nastepnaProba = -1;
            }
            repaint();
        }
    }
}

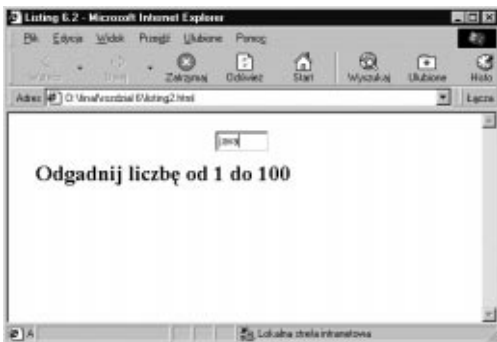
```



Rysunek 6.2. Oto efekt, na który czekaliśmy



Rysunek 6.3. Musisz spróbować jeszcze raz



Rysunek 6.4. Jeśli wpiszesz coś innego niż liczbę, zobaczysz taki komunikat

```
5. if (nastepnaProba < 1) {
    statusLiczby = "Odgadnij liczbę od 1
    ➔ do 100";
}
```

Oto kolejna instrukcja warunkowa if. Tym razem sprawdzamy, czy wartość zmiennej `nastepnaProba` jest mniejsza od 1, co oznacza, że podana wartość jest spoza naszego przedziału. W takim przypadku przypominamy użytkownikowi, jakiej liczby od niego oczekujemy (widać to na rysunku 6.4).

```
6. g.drawString(statusLiczby, 20, 60);
```

Za pomocą tej instrukcji wyświetlamy to, co wcześniej przypisaliśmy wartości `statusLiczby` (odpowiedni komunikat o trafieniu, chybieniu, bądź o podaniu liczby spoza przedziału).

```
7. public void actionPerformed
    ➔ (ActionEvent e) {
    if (e.getSource() instanceof
    ➔ TextField) {
```

Oto metoda `actionPerformed()` wyzwalana zdarzeniem generowanym przez użytkownika. Ten fragment kodu jest uruchamiany za każdym razem, gdy użytkownik wpisuje cokolwiek w polu tekstowym. Na początku metody `actionPerformed()` sprawdzamy, czy zdarzenie, które chcemy obsłużyć, rzeczywiście dotyczy pola tekstowego. Sprawdzamy zatem za pomocą instrukcji warunkowej `if`, czy źródłem zdarzenia jest obiekt pola tekstowego (egzemplarz klasy `TextField`).

Rozdział 6.

```
8. try {  
    nastepnaProba = Integer.parseInt  
        ➤(poleZgadywania.getText());  
}  
catch (NumberFormatException x) {  
    nastepnaProba = -1;  
}  
repaint();
```

Instrukcja try jest specyficznym rodzajem instrukcji warunkowej. W tym przypadku próbujemy wykonać instrukcję, która może spowodować błąd. Umieszczamy ją wewnątrz bloku try...catch. Jeśli jej działanie będzie poprawne (nie spowoduje błędu), nie będzie to miało znaczenia. W przeciwnym wypadku spowoduje wyjątek, który należy przechwycić i obsłużyć (za pomocą instrukcji catch).

W tym przypadku próbujemy przekształcić wpisany przez użytkownika tekst na liczbę. Ponieważ w polu tekstowym użytkownik może wpisać dowolny ciąg znaków, takie przekształcenie może się nie udać i stąd konieczność użycia bloku try...catch.

Jeśli podany łańcuch ma poprawny format liczbowy, przypisujemy wartość otrzymanej liczby zmiennej nastepnaProba.

W przeciwnym razie jest wyzwalany wyjątek NumberFormatException (błędny format liczby), wówczas przypisujemy zmiennej nastepnaProba wartość -1.

Niezależnie od rezultatu przekształcenia wywołujemy metodę repaint(), która przerysuje obszar apletu.

Wskazówka

- Powyższy przykład nie prezentuje najlepszego pomysłu na tego typu grę. Średnio użytkownik musi 50 razy spróbować szczęścia, zanim odgadnie wylosowaną liczbę. W następnym przykładzie zaprezentujemy dużo ciekawsze rozwiązanie.

Java odrzuca nasze polecenia

Wyjątki w Javie można wyrazić przesłaniem od programu mówiącym „zrób to inaczej”. Źródłem wyjątków są nieprzewidziane i niechciane zdarzenia. Przykładowo, Java nie będzie w stanie zamienić liter na liczbę.

Kiedy Java wyzwała wyjątek, Ty, programista, musisz go przechwycić. Zrobisz to za pomocą słowa kluczowego catch, po którym określisz, co program Javy ma zrobić w razie wystąpienia potencjalnych wyjątków.

Listing HTML 6.3. Wyświetla aplet 6.3 w oknie przeglądarki internetowej

```
Listing
<html>
<head>
  <title>Listing 6.3</title>
</head>
<body bgcolor="white">
<object classid="clsid:8AD9C840-044E-11D1-
➤B3E9-00805F499D93" width="500" height="100"
➤codetype="application/java">
  <param name="code" value="Applet3.class">
  <param name="type" value="application/
➤x-java-applet;version=1.3">
  <param name="scriptable" value="false">
  <embed type="application/x-java-applet;
➤version=1.3" code="Applet3.class"
➤width="500" height="100" scriptable="false"
➤pluginspage="http://java.sun.com/products/
➤plugin/1.3/plugin-install.html">
  </embed>
</object>
</body>
</html>
```

Więcej o instrukcjach warunkowych — if...else

Niekiedy niezbędne jest jedynie wyrażenie tego, co ma się stać, jeśli dane wyrażenie jest prawdziwe. Czasem jednak będziesz chciał wyznaczyć instrukcje, które powinny być przetworzone, gdy warunek instrukcji if nie jest spełniony. Służy do tego instrukcja else wskazująca na blok kodu uruchamiany tylko wtedy, gdy warunek instrukcji warunkowej if nie jest spełniony. Listing HTML 6.3 i aplet 6.3 prezentują poprawiony aplet gry „zgadnij, jaką sobie wymyśliłem liczbę”, który prezentuje użytkownikowi kierunek, w którym powinien podążać.

Aby korzystać z konstrukcji if...else, należy zastosować:

1. if (następnaProba != szukanaLiczba) {

Zaczynamy od sprawdzenia, czy podana przez użytkownika liczba różni się od wylosowanej. Fragment kodu poprzedzony tą instrukcją warunkową zostanie wykonany tylko w przypadku niezgodności liczby użytkownika z wylosowaną przez aplet.

2. if (następnaProba < 1) {
 statusLiczby = "Odgadnij liczbę od 1
 ➤do 100";
 }

Teraz sprawdzamy, czy zmienna następnaProba ma wartość -1, co oznaczałoby, że metoda paint() działa po raz pierwszy lub że użytkownik wpisał ciąg znaków, którego nie można przekształcić na liczbę. Jeśli warunek jest spełniony, wyświetlamy komunikat „Odgadnij liczbę od 1 do 100”.

3. else {

W przeciwnym wypadku, jeśli wartość zmiennej `nastepnaProba` jest różna od wartości zmiennej `szukanaLiczba` oraz `nastepnaProba` jest poprawną liczbą, uruchamiamy fragment kodu poprzedzony słowem `else`. W tym przypadku fragment składa się z kolejnego bloku `if...else`.

4. `if (nastepnaProba < szukanaLiczba) {`
 `statusLiczby = nastepnaProba + "`
 `to za mało";`
 `}`
 `else {`
 `statusLiczby = nastepnaProba + "`
 `to za dużo";`
 `}`

Teraz sprawdzamy, czy wartość zmiennej `nastepnaProba` jest mniejsza od wartości przechowywanej w zmiennej `szukanaLiczba`. Jeśli tak, zmienna `nastepnaProba` będzie wyświetlona z komunikatem, że podana liczba jest za mała. W przeciwnym razie zakomunikujemy użytkownikowi, że jest za duża. Na rysunku 6.5 prezentujemy wygląd apletu, gdy podana liczba jest za duża, rysunki 6.6 i 6.7 prezentują efekt dla zbyt małej liczby. Jeśli w końcu podamy liczbę równą szukanej, zobaczymy komunikat widoczny na rysunku 6.8.

Aplet 6.3. Ulepszona wersja gry „zgadnij, jaką sobie wymyśliłem liczbę”

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet3 extends Applet implements
    ActionListener {
    TextField poleZgadywania = new TextField(5);
    int nastepnaProba = -1;
    int szukanaLiczba = (int)(java.lang.Math.
        random() * 100) + 1;
    Font f = new Font("TimesRoman", Font.BOLD,
        24);

    public void init() {
        setBackground(Color.white);
        poleZgadywania.addActionListener(this);
        add(poleZgadywania);
    }

    public void paint(Graphics g) {
        String statusLiczby = nastepnaProba + "
            jest szukaną liczbą";

        g.setFont(f);
        if (nastepnaProba != szukanaLiczba) {
            if (nastepnaProba < 1) {
                statusLiczby = "Odgadnij liczbę od 1
                    do 100";
            }
            else {
                if (nastepnaProba < szukanaLiczba) {
                    statusLiczby = nastepnaProba + " to
                        za mało";
                }
                else {
                    statusLiczby = nastepnaProba + " to
                        za dużo";
                }
            }
        }
        g.drawString(statusLiczby, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                nastepnaProba = Integer.parseInt
                    (poleZgadywania.getText());
            }
            catch (NumberFormatException x) {
                nastepnaProba = -1;
            }
            repaint();
        }
    }
}

```



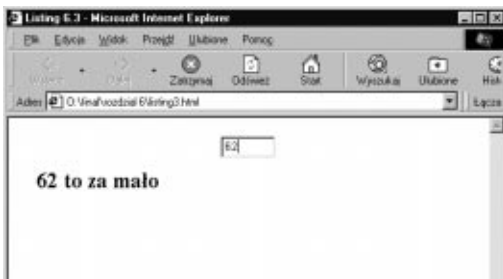

Rysunek 6.5. Zaczniemy od liczby 50 i zobaczymy, czy jest za duża czy za mała

Wskazówka

- W poprzednim przypadku użytkownik musiał próbować 50 razy, by osiągnąć 50% szans na sukces. W tym przypadku mamy taką samą możliwość odgadnięcia liczby po najwyżej siedmiu próbach.



Rysunek 6.6. Połowa z różnicy między 50 a 100 to 75 — okazuje się, że to za dużo



Rysunek 6.7. Połowa z różnicy między 50 a 75 to 62 — tym razem za mało



Rysunek 6.8. Tym razem podajemy 71 — nareszcie, prawidłowa wartość

Powtarzanie czynności za pomocą pętli

Struktura `if...else`, którą posługiwaliśmy się w poprzednim przykładzie, pozwalała uruchamiać albo jeden, albo drugi blok kodu. W przyszłości często możesz stawiać przed koniecznością wpisania jednej lub wielu instrukcji, które powinny być uruchomione wielokrotnie. Rozwiązaniem, które pozwala to zrealizować, jest *pętla*. Listing HTML 6.4 i aplet 6.4 demonstrują jeden z rodzajów pętli: `for`. W poniższym przykładzie umożliwimy użytkownikowi wpisanie liczby, na podstawie której wyświetlimy rząd gwiazdek w oknie przeglądarki.

Zapętlenie kodu

Pętla `for` składa się z trzech części (patrz rysunek 6.10):

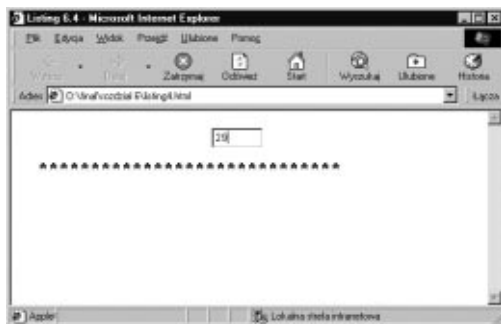
1. *Inicjalizacja* — podczas pierwszego przejścia pętli inicjalizowana jest zmienna pętli — *licznik pętli*.
2. *Ograniczenie* — tak określamy, kiedy pętla się zakończy. W normalnych warunkach ludzie odliczają od jednego do dziesięciu; powszechną praktyką w językach programowania jest odliczanie od zera do dziewięciu. W obu przypadkach kod wewnątrz pętli zostanie przetworzony dziesięć razy, jednak metoda liczenia od zera jest wygodniejsza w językach programowania takich jak Java z racji indeksowania tablic od zera. Dlatego właśnie określamy ograniczenia w pętli jako „mniejszy niż wartość `liczbaUzytkownika`”, a nie „mniejszy lub równy wartości `liczbaUzytkownika`”.
3. *Inkrementacja* — w tej części określamy, jak dalece zwiększać licznik pętli w każdej kolejnej iteracji pętli. W tym przypadku dodajemy jeden za pomocą znaków `++` po identyfikatorze zmiennej.

<code>i=0;</code>	<code>i<liczbaUzytkownika;</code>	<code>i++</code>
Inicjalizacja	Ograniczenie	Inkrementacja

Rysunek 6.10. Trzy części pętli

Listing HTML 6.4. Oto, jak możemy wywołać nasz nowy *aplet*

```
Listing
<html>
<head>
  <title>Listing 6.4</title>
</head>
<body bgcolor="white">
  <object classid="clsid:8AD9C840-044E-11D1-
  ➤B3E9-00805F499D93" width="500" height="100"
  ➤codetype="application/java">
    <param name="code" value="Applet4.class">
    <param name="type" value="application/
    ➤x-java-applet;version=1.3">
    <param name="scriptable" value="false">
    <embed type="application/x-java-applet;
    ➤version=1.3" code="Applet4.class" width=
    ➤"500" height="100" scriptable="false"
    ➤pluginspage="http://java.sun.com/products/
    ➤plugin/1.3/plugin-install.html">
  </embed>
</object>
</body>
</html>
```



Rysunek 6.9. Użytkownik poprosił o 29 gwiazdek i natychmiast je otrzymał

Aplet 6.4. Aplet „pyta” użytkownika o liczbę gwiazdek i wyświetla je za pomocą pętli

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet4 extends Applet implements
↳ ActionListener {
    TextField poleUzytkownika = new TextField(5);
    int liczbaUzytkownika = -1;
    Font f = new Font("TimesRoman", Font.BOLD,
↳ 24);

    public void init() {
        setBackground(Color.white);
        poleUzytkownika.addActionListener(this);
        add(poleUzytkownika);
    }

    public void paint(Graphics g) {
        String lancuchWyjsciuowy = "";
        int i;

        g.setFont(f);
        if (liczbaUzytkownika < 1 ||
↳ liczbaUzytkownika > 50) {
            lancuchWyjsciuowy = "Podaj liczbę od 1
↳ do 50";
        }
        else {
            for (i=0; i<liczbaUzytkownika; i++) {
                lancuchWyjsciuowy = lancuchWyjsciuowy +
↳ " *";
            }
        }
        g.drawString(lancuchWyjsciuowy, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                liczbaUzytkownika = Integer.parseInt
↳ (poleUzytkownika.getText());
            }
            catch (NumberFormatException x) {
                liczbaUzytkownika = -1;
            }
            repaint();
        }
    }
}

```

Aby korzystać z pętli for, należy zastosować:

1. if (liczbaUzytkownika < 1 ||
↳ liczbaUzytkownika > 50) {
 lancuchWyjsciuowy = "Podaj liczbę od 1
↳ do 50";
}
2. else {
 for (i=0; i<liczbaUzytkownika; i++) {
 lancuchWyjsciuowy = lancuchWyjsciuowy
↳ + " *";
 }
}

Jeśli liczba jest prawidłowa (nie spełniony warunek instrukcji if z poprzedniego kroku), przechodzimy do pętli for. Pierwsza część inicjalizuje licznik pętli i wartością 0. W drugiej części określamy, że pętla będzie działać tak długo, aż i będzie mniejsze od wartości podanej przez użytkownika. Trzeci element określa operację wykonywaną przy każdej iteracji pętli, w tym przypadku licznik pętli będzie zwiększany o 1. Na rysunku 6.9 widać efekt działania apletu po tym, jak użytkownik wpisał „29”. Do zmiennej lancuchWyjsciuowy (początkowo jest pustym łańcuchem) dodajemy gwiazdkę przy każdym nawrocie pętli.

Wskazówka

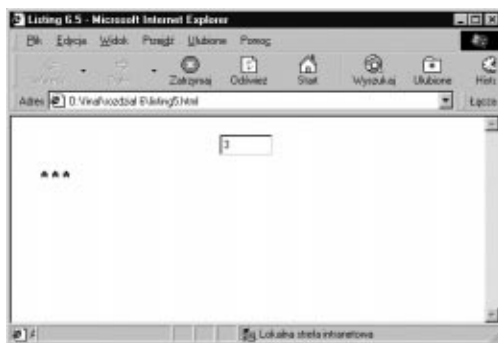
- Znaki || pomiędzy warunkami sprawdzającymi, czy liczbaUzytkownika jest mniejsza od 1 i większa niż 50 oznaczają logiczną alternatywę (czyli „lub”). Mówi to, że cały warunek if jest prawdziwy, jeśli choć jeden z warunków składowych (rozdzielonych ||) jest prawdziwy. Gdybyśmy chcieli, by konieczna była prawdziwość oby warunków, rozdzielilibyśmy je znakami && oznaczającymi koniunkcję (czyli „i”).

Inny rodzaj pętli — while

W Javie istnieją trzy sposoby budowania pętli. Pierwszym jest pętla omówiona w poprzednim przykładzie — pętla `for`, dwie pozostałe to odmiany pętli `while`. Kod zawarty w pętli `while` działa tak długo, jak długo spełniony jest warunek tej pętli (patrz listing HTML 6.5 i aplet 6.5). Przy pierwszym sprawdzeniu tego warunku, które wykaże jego nieprawdziwość, działanie pętli jest kończone. Jeśli już przy pierwszym sprawdzeniu, warunek okaże się nieprawdziwy, pętla w ogóle nie zostanie uruchomiona.

Listing HTML 6.5. Wstawia aplet 6.5 na stronę WWW

```
Listing
<html>
<head>
  <title>Listing 6.5</title>
</head>
<body bgcolor="white">
<object classid="clsid:8AD9C840-044E-11D1-
  B3E9-00805F499D93" width="500" height="100"
  >codetype="application/java">
  <param name="code" value="Applet5.class">
  <param name="type" value="application/
    x-java-applet;version=1.3">
  <param name="scriptable" value="false">
  <embed type="application/x-java-applet;
    version=1.3" code="Applet5.class" width=
    "500" height="100" scriptable="false"
    pluginspage="http://java.sun.com/products/
    plugin/1.3/plugin-install.html">
  </embed>
</object>
</body>
</html>
```



Rysunek 6.11. Teraz użytkownik poprosił jedynie o trzy gwiazdki

Aplet 6.5. Ciekawe, ile gwiazdek użytkownik będzie chciał wyświetlić tym razem

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet5 extends Applet
    implements ActionListener {
    TextField poleUzytkownika = new TextField(5);
    int liczbaUzytkownika = -1;
    Font f = new Font("TimesRoman", Font.BOLD,
        24);

    public void init() {
        setBackground(Color.white);
        poleUzytkownika.addActionListener(this);
        add(poleUzytkownika);
    }

    public void paint(Graphics g) {
        String lancuchWyjsciuowy = "";
        int i = 0;

        g.setFont(f);
        if (liczbaUzytkownika > 50) {
            liczbaUzytkownika = -1;
        }
        if (liczbaUzytkownika < 1) {
            lancuchWyjsciuowy = "Podaj liczbę od 1
                do 50";
        }
        while (i < liczbaUzytkownika) {
            lancuchWyjsciuowy = lancuchWyjsciuowy + "*";
            i++;
        }
        g.drawString(lancuchWyjsciuowy, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                liczbaUzytkownika = Integer.parseInt
                    (poleUzytkownika.getText());
            }
            catch (NumberFormatException x) {
                liczbaUzytkownika = -1;
            }
            repaint();
        }
    }
}

```

Aby używać pętli while, należy wpisać:

```

1. while (i < liczbaUzytkownika)
    lancuchWyjsciuowy = lancuchWyjsciuowy +
        " *";
    i++;
}

```

Wartość zmiennej `liczbaUzytkownika` albo mieści się w przedziale od 1 do 50 (wówczas jest prawidłowa), albo wynosi -1 (jeśli użytkownik wpisał niepoprawny łańcuch). Ponieważ `i` jest inicjalizowane wartością 0, powyższa pętla w ogóle nie zadziała, jeśli nie podano prawidłowej wartości, ponieważ -1 nie jest większe od 0. Jeśli podana przez użytkownika wartość jest prawidłowa, kod zawarty w pętli zostanie przetworzony tyle razy, ile przypisano zmiennej `liczbaUzytkownika`. Za każdym razem zmienna `i` jest zwiększana o 1. Przykładowo, jeśli wartość zmiennej `liczbaUzytkownika` wynosi 25, pętla uruchomi się 25 razy. Jeśli `liczbaUzytkownika` wynosi -1, pętla `while` przed pierwszym uruchomieniem stwierdzi nieprawdziwość warunku `i < liczbaUzytkownika` i nie uruchomi się ani razu.

Wskazówka

- Upewnij się, że pętla `while` kiedyś się zakończy. Gdybyś, przykładowo, pominął instrukcję `i++` w powyższym przykładzie, stworzył byś nieskończoną pętlę, czyli taką, która sama nigdy się nie zakończy.

Ostatni rodzaj pętli — do...while

Trzecim i ostatnim typem pętli w Javie jest do...while. W układzie do...while sprawdzamy warunek pętli na końcu pętli, a nie na początku. Oznacza to, że pętla zawsze się uruchomi przynajmniej raz. Listing HTML 6.6 i aplet 6.6 prezentują zastosowanie pętli do...while.

Aby użyć pętli do...while, należy wpisać:

```
1. do {
    lancuchWyjsciuowy = lancuchWyjsciuowy +
        "\n*";
    i++;
} while (i < liczbaUzytkownika);
```

W takim układzie zmienna łańcuchowa lancuchWyjsciuowy będzie zawsze zawierała przynajmniej jedną gwiazdkę. Musimy więc uważać, w jaki sposób umieścimy ten fragment w naszym programie — w tym przypadku sprawdzamy przed wejściem do pętli, czy wartość zmiennej liczbaUzytkownika jest poprawna (od 1 do 50).

Pętla będzie działać tak długo, jak długo i będzie mniejsze od liczbaUzytkownika. Efekt widać na rysunku 6.12.

Listing HTML 6.6. Poniższy plik HTML wywołuje aplet 6.6 zawierający pętlę do...while

```
Listing
<html>
<head>
  <title>Listing 6.6</title>
</head>
<body bgcolor="white">
  <object classid="clsid:8AD9C840-044E-11D1-
    B3E9-00805F499D93" width="500" height="100"
    codetype="application/java">
    <param name="code" value="Applet6.class">
    <param name="type" value="application/
      x-java-applet;version=1.3">
    <param name="scriptable" value="false">
    <embed type="application/x-java-applet;
      version=1.3" code="Applet6.class" width=
      "500" height="100" scriptable="false"
      pluginspage="http://java.sun.com/products/
      plugin/1.3/plugin-install.html">
    </embed>
  </object>
</body>
</html>
```



Rysunek 6.12. Użytkownik zażądał 34 gwiazdki

Aplet 6.6. *Za pomocą pętli do...while przetwarzamy żądanie użytkownika*

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet6 extends Applet implements
↳ ActionListener {
    TextField poleUzytkownika = new TextField(5);
    int liczbaUzytkownika = -1;
    Font f = new Font("TimesRoman", Font.BOLD,
↳ 24);

    public void init() {
        setBackground(Color.white);
        poleUzytkownika.addActionListener(this);
        add(poleUzytkownika);
    }

    public void paint(Graphics g) {
        String lancuchWyjsciowy = "";
        int i = 0;

        g.setFont(f);
        if (liczbaUzytkownika < 1 ||
↳ liczbaUzytkownika > 50) {
            lancuchWyjsciowy = "Podaj liczbę od 1
↳ do 50";
        }
        else {
            do {
                lancuchWyjsciowy = lancuchWyjsciowy +
↳ "*";
                i++;
            } while (i < liczbaUzytkownika);
        }
        g.drawString(lancuchWyjsciowy, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                liczbaUzytkownika = Integer.parseInt
↳ (poleUzytkownika.getText());
            }
            catch (NumberFormatException x) {
                liczbaUzytkownika = -1;
            }
            repaint();
        }
    }
}

```

Przerywanie działania pętli

Istnieje możliwość opuszczenia pętli przetwarzanej zgodnie z jej warunkami. Taki sposób kończenia pętli można uzyskać za pomocą instrukcji `break`.

W listingu HTML 6.7 i aplecie 6.7 użytkownik może wpisać dowolną liczbę. Niezależnie jednak od jej wielkości, pętla nie zadziała więcej niż dwadzieścia razy.

Aby przerwać działanie pętli, należy użyć:

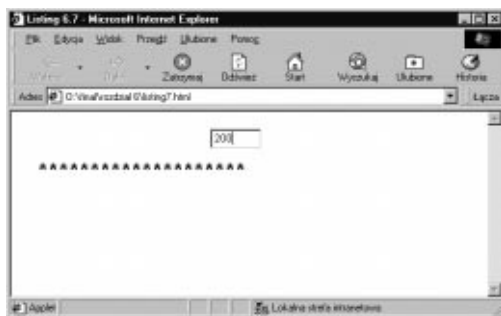
```
1. for (i=0; i<liczbaUzytkownika; i++) {
    if (i >= 20) break;
    lancuchWyjsciowy = lancuchWyjsciowy +
    ➤ " * ";
}
```

To jest standardowa pętla `for`, która w najprostszej formie działałaby aż do osiągnięcia przez licznik `i` wartości zmiennej `liczbaUzytkownika`. Wstawienie instrukcji `break` wewnątrz pętli sprawi, że jej działanie zostanie przerwane, gdy `i` osiągnie wartość 20.

Instrukcja `break` powoduje skok do pierwszej instrukcji znajdującej się za pętlą — w naszym przypadku jest to `g.drawString(lancuchWyjsciowy, 20, 60)`. Efekt jest widoczny na rysunku 6.13.

Listing HTML 6.7. Wstawia na stronę aplet 6.7

```
Listing
<html>
<head>
  <title>Listing 6.7</title>
</head>
<body bgcolor="white">
<object classid="clsid:8AD9C840-044E-11D1-
➤ B3E9-00805F499D93" width="500" height="100"
➤ codetype="application/java">
  <param name="code" value="Applet7.class">
  <param name="type" value="application/
➤ x-java-applet;version=1.3">
  <param name="scriptable" value="false">
  <embed type="application/x-java-applet;
➤ version=1.3" code="Applet7.class" width=
➤ "500" height="100" scriptable="false"
➤ pluginspage="http://java.sun.com/products/
➤ plugin/1.3/plugin-install.html">
  </embed>
</object>
</body>
</html>
```



Rysunek 6.13. Niezależnie od tego, jaką liczbę podamy, zobaczymy maksymalnie 20 gwiazdek

Aplet 6.7. *Demonstruje opuszczenie pętli za pomocą instrukcji break*

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet7 extends Applet implements
↳ ActionListener {
    TextField poleUzytkownika = new TextField(5);
    int liczbaUzytkownika = -1;
    Font f = new Font("TimesRoman", Font.BOLD,
↳ 24);

    public void init() {
        setBackground(Color.white);
        poleUzytkownika.addActionListener(this);
        add(poleUzytkownika);
    }

    public void paint(Graphics g) {
        String lancuchWyjsciowy = "";
        int i;

        g.setFont(f);
        if (liczbaUzytkownika < 1) {
            lancuchWyjsciowy = "Podaj liczbę od 1
↳ do 20";
        }
        else {
            for (i=0; i<liczbaUzytkownika; i++) {
                if (i >= 20) break;
                lancuchWyjsciowy = lancuchWyjsciowy +
↳ "*";
            }
        }
        g.drawString(lancuchWyjsciowy, 20, 60);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof TextField) {
            try {
                liczbaUzytkownika = Integer.parseInt
↳ (poleUzytkownika.getText());
            }
            catch (NumberFormatException x) {
                liczbaUzytkownika = -1;
            }
            repaint();
        }
    }
}

```

Wskazówka

- W Javie istnieje także instrukcja `continue`. Kiedy jej użyjemy, następujący po niej fragment kodu wewnątrz pętli zostanie pominięty, nie przerywamy jednak jej działania. W powyższym przypadku, gdybyśmy zastąpili instrukcję `break` instrukcją `continue`, pomijane byłyby instrukcje dopisywania gwiazdki do zmiennej `lancuchWyjsciowy` od momentu osiągnięcia przez `i` wartości 20. W przeciwieństwie jednak do przerywania działania pętli instrukcją `break`, gdybyśmy użyli `continue`, pętla działałaby nadal, aż do momentu zrównania się licznika `i` z wartością zmiennej `liczbaUzytkownika`.

Konstrukcja switch...case

Java oferuje jeszcze jeden sposób kontrolowania przetwarzania kodu — konstrukcję switch...case. Najprostszym sposobem zrozumienia istoty tej instrukcji jest spojrzenie na nią jak na zbiór instrukcji if...else. Zamiast powiedzieć: „jeśli dzisiaj jest poniedziałek, zrób to; w przeciwnym razie, jeśli dzisiaj jest wtorek, zrób coś innego; w przeciwnym razie, jeśli dzisiaj jest środa, zrób coś jeszcze innego itd.”, możemy użyć konstrukcji switch...case, by bezpośrednio wyróżnić każdą z możliwości.

Listing HTML 6.8 i aplet 6.8 prezentują grę w *kamień, nożyce i papier* napisaną z użyciem konstrukcji switch...case.

Aby użyć instrukcji switch...case, należy zastosować:

1. `public class Applet8 extends Applet`
`↳ implements ItemListener {`
 Ponieważ w naszym nowym aplecie do interakcji z użytkownikiem posłużymy się przyciskami opcji zamiast polem tekstowym, musimy użyć zupełnie innych metod do przechwytywania zdarzeń. Korzystamy więc z interfejsu `ItemListener`, który dostarcza nam metodę `itemStateChanged()` wyzwalaną przez zdarzenie kliknięcia przycisku opcji.
2. `CheckboxGroup userCheckbox;`
`Checkbox kamienCheckBox, nozyceCheckBox,`
`↳ papierCheckBox;`
 Każdy przycisk opcji `Checkbox` jest początkowo definiowany jako pole wyboru. Później, w kroku 4., określimy, że wszystkie trzy pola będą przyciskami opcji należącymi do jednej grupy `CheckboxGroup`.
3. `userCheckbox = new CheckboxGroup();`
 Tworzymy obiekt klasy `CheckboxGroup` (grupa pól wyboru) nazwany `userCheckbox`.

Listing HTML 6.8. Tak wstawiamy naszą grę zapisaną w aplecie 6.8

```
Listing
<html>
<head>
  <title>Listing 6.8</title>
</head>
<body bgcolor="white">
  <object classid="clsid:8AD9C840-044E-11D1-
  ↳B3E9-00805F499D93" width="500" height="100"
  ↳codetype="application/java">
    <param name="code" value="Applet8.class">
    <param name="type" value="application/
    ↳x-java-applet;version=1.3">
    <param name="scriptable" value="false">
    <embed type="application/x-java-applet;
    ↳version=1.3" code="Applet8.class" width=
    ↳"500" height="100" scriptable="false"
    ↳pluginspage="http://java.sun.com/products/
    ↳plugin/1.3/plugin-install.html">
    </embed>
  </object>
</body>
```

ch...case

Aplet 6.8. Zobaczmy, czy możemy wygrać z naszym apletem

```

Aplet
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Aplet8 extends Applet implements
    ItemListener {
    int liczbaApletu = (int)(java.lang.Math.
        random() * 3) + 1;
    Font f = new Font("TimesRoman", Font.BOLD,
        16);
    CheckboxGroup userCheckbox;
    Checkbox kamienCheckBox, nozyceCheckBox,
        papierCheckBox;
    boolean pierwszaProba = true;
    String wyborUzytkownika;

    public void init() {
        setBackground(Color.white);

        userCheckbox = new CheckboxGroup();
        kamienCheckBox = new Checkbox("Kamień",
            userCheckbox, false);
        kamienCheckBox.addItemListener(this);
        add(kamienCheckBox);

        nozyceCheckBox = new Checkbox("Nożyce",
            userCheckbox, false);
        nozyceCheckBox.addItemListener(this);
        add(nozyceCheckBox);

        papierCheckBox = new Checkbox("Papier",
            userCheckbox, false);
        papierCheckBox.addItemListener(this);
        add(papierCheckBox);
    }

    public void paint(Graphics g) {
        int liczbaUzytkownika;
        String wyborApletu;

        g.setFont(f);

        if (pierwszaProba) {
            g.drawString("Zagraj ze mną w kamień,
                nożyce i papier!", 20, 60);
            pierwszaProba = false;
        }
        else {
            switch(liczbaApletu) {
                case 1:
                    wyborApletu = "Kamień";
                    break;
                case 2:
                    wyborApletu = "Nożyce";
                    break;
            }
        }
    }
}

```

4. kamienCheckBox = new Checkbox("Kamień",
 ➔userCheckbox, false);
 kamienCheckBox.addItemListener(this);
 ➔add(kamienCheckBox);

Te trzy instrukcje tworzą przycisk opcji kamienCheckBox będący częścią grupy userCheckbox. Klikając którykolwiek przycisk należący do tej grupy, spowodujesz, że wszystkie pozostałe przyciski z tej grupy zostaną wyłączone. Pierwszym parametrem metody Checkbox() jest *etykieta* (nazwa przycisku wypisana na ekranie), drugim parametrem jest wspomniana grupa przycisków, jako trzeci parametr podajemy stan przycisku (true lub false, zależnie od tego, czy chcemy, by był na początku włączony czy wyłączony). Następnie ustawiamy addItemListener(this) dla nowego przycisku opcji, co pozwoli nam dalej przechwytywać zdarzenia związane z przyciskiem za pomocą metody itemStateChanged(). Kończymy ten fragment kodu, wstawiając nasz przycisk opcji w obszarze apletu.

```

5. switch(liczbaApletu) {
    case 1:
        wyborApletu = "Kamień";
        break;
    case 2:
        wyborApletu = "Nożyce";
        break;
    case 3:
        wyborApletu = "Papier";
        break;
    default:
        wyborApletu = "Błąd";
}

```

Podobnie jak we wcześniejszych przykładach, użyliśmy generatora liczb losowych Javy do przypadkowego wybrania kamienia, nożyc lub papieru przez nasz aplet. W efekcie zmienna liczbaApletu zawiera liczbę losową od 1 do 3. Teraz instrukcja switch() wybiera odpowiedni blok kodu case w zależności od wartości zmiennej liczbaApletu. Powyższy fragment moglibyśmy zapisać następująco:

```

if (liczbaApletu==1) {
    wyborApletu = "Kamień";
}
else
    if (liczbaApletu==2) {
        wyborApletu = "Nożyce";
    }
    else
        if...

```

I tak dalej, dla każdej możliwej wartości.

```

6. switch (wyborUzytkownika.charAt(0)) {
    case 'K':
        liczbaUzytkownika = 1;
        break;
    case 'N':
        liczbaUzytkownika = 2;
        break;
    case 'P':
        liczbaUzytkownika = 3;
        break;
    default:
        liczbaUzytkownika = 0;
}

```

W tym fragmencie kodu sprawdzamy zmienną typu char (podobny typ do łańcucha, tyle że przechowujący tylko jeden znak), a nie, jak poprzednio, liczbę. Użytkownik dokonuje wyboru, klikając przycisk opcji, co wyzwała zdarzenie Javy. Możemy odczytać etykietę klikniętego przycisku i przypisać zmiennej całkowitoliczbowej `liczbaUzytkownika` odpowiednią wartość w celu późniejszego porównania z wyborem apletu (zmienna `liczbaApletu`). Rysunek 6.14 prezentuje widok apletu na początku gry. Rysunki 6.15, 6.16 i 6.17 prezentują możliwe wyniki.

```

7. public void itemStateChanged(ItemEvent e) {
    if (e.getSource() instanceof Checkbox) {
        wyborUzytkownika = userCheckbox.
            ▶getSelectedCheckbox().getLabel();
    }
}

```

Oto fragment kodu wyzwalany kliknięciem myszą przez użytkownika. Jeśli obiektem interakcji był przycisk opcji, zmiennej `wyborUzytkownika` przypisujemy etykietę tego przycisku.

Aplet 6.8. Zobaczmy, czy możemy wygrać z naszym apletem — ciąg dalszy

```

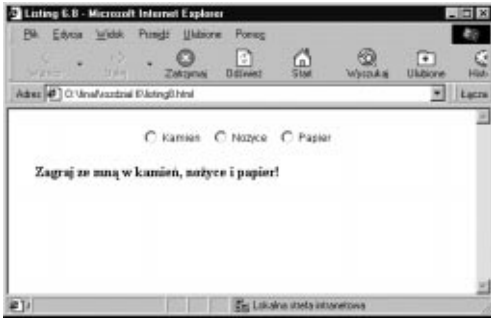
Aplet
case 3:
    wyborApletu = "Papier";
    break;
default:
    wyborApletu = "Błąd";
}

switch (wyborUzytkownika.charAt(0)) {
    case 'K':
        liczbaUzytkownika = 1;
        break;
    case 'N':
        liczbaUzytkownika = 2;
        break;
    case 'P':
        liczbaUzytkownika = 3;
        break;
    default:
        liczbaUzytkownika = 0;
}

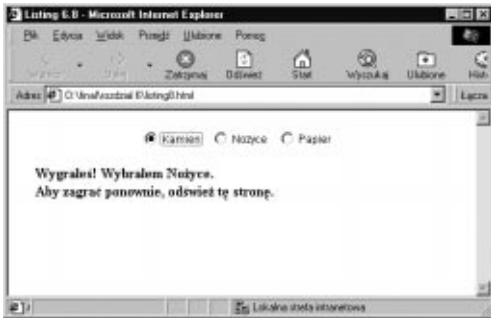
if (liczbaApletu == liczbaUzytkownika) {
    g.drawString("REMIS - zagrajmy jeszcze
        ▶raz.", 20, 60);
}
else {
    if ((liczbaUzytkownika==1 &&
        ▶liczbaApletu==3) ||
        ▶(liczbaUzytkownika==2 &&
        ▶liczbaApletu==1) ||
        ▶(liczbaUzytkownika==3 &&
        ▶liczbaApletu==2)) {
        g.drawString("Wygrałem! Wybrałem "
            ▶+ wyborApletu + ".", 20, 60);
    }
    else {
        g.drawString("Wygrałeś! Wybrałem "
            ▶+ wyborApletu + ".", 20, 60);
    }
}
g.drawString("Aby zagrać ponownie,
    ▶odśwież tę stronę.", 20, 80);
}

public void itemStateChanged(ItemEvent e) {
    if (e.getSource() instanceof Checkbox) {
        wyborUzytkownika = userCheckbox.
            ▶getSelectedCheckbox().getLabel();
        repaint();
    }
}
}
}

```



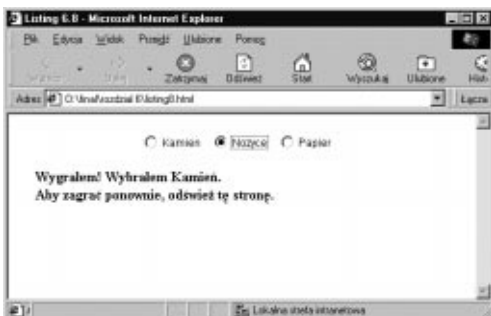
Rysunek 6.14. Aplet jest gotowy do nowej gry



Rysunek 6.15. Użytkownik wygrał pierwszą grę



Rysunek 6.16. W drugiej grze nastąpił remis



Rysunek 6.17. Ostatnią grę wygrał komputer

Wskazówki

- Instrukcja `switch...case` może być używana jedynie dla czterech typów prymitywów: `byte`, `char`, `int` i `short`. Właśnie dlatego w szóstym kroku sprawdzamy jedynie pierwszą literę zmiennej łańcuchowej `wyborUzytkownika` zamiast całego łańcucha.
- Gdybyśmy pominęli którąkolwiek instrukcję `break` w powyższych blokach `case`, przetwarzane byłyby także następne bloki `case`. Przerwanie wykonania instrukcji `switch...case` jest bardzo przydatne, może jednak być źródłem wielu błędów w oprogramowaniu. Pamiętaj więc o wstawianiu instrukcji `break` na końcu każdego bloku `case`.
- Jeśli żadna z wartości wymienionych w instrukcjach `case` nie pasuje do wzorca, przetwarzany jest blok domyślny (rozpoczynający się od słowa `default`). Nawet kiedy jesteś przekonany o tym, że któraś z opcji musi zostać wybrana, dla pewności używaj bloku `default`.