

MARCIN LIS

WYDANIE IV

Java



**PRAKTYCZNY
KURS**

Helion 

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

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)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/pkjav4>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Listingi wykorzystane w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/pkjav4.zip>

ISBN: 978-83-246-9663-5

Copyright © Helion 2015

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wstęp	5
Rozdział 1. Podstawy	9
Instalacja JDK	9
Instalacja w systemie Windows	10
Instalacja w systemie Linux	11
Przygotowanie do pracy z JDK	11
Podstawy programowania	13
Lekcja 1. Struktura programu, kompilacja i wykonanie	13
Lekcja 2. Podstawy obiektowości i typy danych	16
Lekcja 3. Komentarze	19
Rozdział 2. Instrukcje języka	23
Zmienne	23
Lekcja 4. Deklaracje i przypisania	24
Lekcja 5. Wyprowadzanie danych na ekran	27
Lekcja 6. Operacje na zmiennych	33
Instrukcje sterujące	47
Lekcja 7. Instrukcja warunkowa if...else	47
Lekcja 8. Instrukcja switch i operator warunkowy	53
Lekcja 9. Pętle	59
Lekcja 10. Instrukcje break i continue	66
Tablice	74
Lekcja 11. Podstawowe operacje na tablicach	74
Lekcja 12. Tablice wielowymiarowe	79
Rozdział 3. Programowanie obiektowe. Część I	91
Podstawy	91
Lekcja 13. Klasy, pola i metody	91
Lekcja 14. Argumenty i przeciążanie metod	100
Lekcja 15. Konstruktory	110
Dziedziczenie	121
Lekcja 16. Klasy potomne	122
Lekcja 17. Specyfikatory dostępu i pakiety	129
Lekcja 18. Przesłanianie metod i składowe statyczne	144
Lekcja 19. Klasy i składowe finalne	156

Rozdział 4. Wyjątki	165
Lekcja 20. Blok try... catch	165
Lekcja 21. Wyjątki to obiekty	173
Lekcja 22. Własne wyjątki	182
Rozdział 5. Programowanie obiektowe. Część II	195
Polimorfizm	195
Lekcja 23. Konwersje typów i rzutowanie obiektów	195
Lekcja 24. Późne wiązanie i wywoływanie metod klas pochodnych	204
Lekcja 25. Konstruktory oraz klasy abstrakcyjne	212
Interfejsy	222
Lekcja 26. Tworzenie interfejsów	222
Lekcja 27. Wiele interfejsów	230
Klasy wewnętrzne	238
Lekcja 28. Klasa w klasie	238
Lekcja 29. Rodzaje klas wewnętrznych i dziedziczenie	246
Lekcja 30. Klasy anonimowe i zagnieżdżone	254
Rozdział 6. System wejścia-wyjścia	269
Lekcja 31. Standardowe wejście	269
Lekcja 32. Standardowe wejście i wyjście	279
Lekcja 33. System plików	295
Lekcja 34. Operacje na plikach	306
Rozdział 7. Kontenery i typy uogólnione	323
Lekcja 35. Kontenery	323
Lekcja 36. Typy uogólnione	336
Rozdział 8. Aplikacje i aplety	351
Aplety	351
Lekcja 37. Podstawy apletów	351
Lekcja 38. Kroje pisma (fonty) i kolory	358
Lekcja 39. Grafika	366
Lekcja 40. Dźwięki i obsługa myszy	376
Aplikacje	386
Lekcja 41. Tworzenie aplikacji	386
Lekcja 42. Komponenty	402
Skorowidz	417

Rozdział 6.

System wejścia-wyjścia

Do pisania aplikacji w Javie niezbędna jest znajomość przynajmniej podstaw systemu wejścia-wyjścia. Tej tematyce jest poświęcony ten rozdział. W Javie operacje wejścia-wyjścia są wykonywane za pomocą strumieni. Strumień to abstrakcyjny ciąg danych, który działa — mówiąc w uproszczeniu — w taki sposób, że dane wprowadzone na jednym jego końcu pojawiają się na drugim końcu. Strumienie dzielą się na wejściowe i wyjściowe. Strumienie wyjściowe mają początek w aplikacji i koniec w innym urządzeniu, np. na ekranie czy w pliku, umożliwiają zatem wyprowadzanie danych z programu. Przykładowo standardowy strumień wyjściowy jest reprezentowany przez obiekt `System.out`, a wykonanie metody `println` tego obiektu powoduje, że dane domyślnie zostają wysłane na ekran. Strumienie wejściowe działają odwrotnie: ich początek znajduje się poza aplikacją (może to być np. klawiatura albo plik dyskowy), a koniec w aplikacji, umożliwiają zatem wprowadzanie danych. Co więcej, strumienie umożliwiają też komunikację między obiektami w obrębie jednej aplikacji. W tym rozdziale zostanie jednak omówiona tylko komunikacja aplikacji ze światem zewnętrznym.

Lekcja 31. Standardowe wejście

Opisy podstawowych operacji wyjściowych, czyli wyświetlania informacji na ekranie konsoli, pojawiły się już wielokrotnie; operacje te zostaną dokładniej przedstawione w ramach kolejnej lekcji. Omówienie systemu wejścia-wyjścia rozpoczniemy od wprowadzania danych i standardowego strumienia wejściowego reprezentowanego przez obiekt `System.in`. Tej tematyce zostanie poświęcona cała bieżąca lekcja. Zostaną w niej przedstawione informacje, jak odczytywać dane wprowadzane przez użytkownika z klawiatury.

Standardowy strumień wejściowy

Standardowy strumień wejściowy jest reprezentowany przez obiekt `System.in`, czyli obiekt `in` zawarty w klasie `System`. Jest to obiekt typu `InputStream`, klasy reprezentującej strumień wejściowy. Metody udostępniane przez tę klasę są zebrane w tabeli 6.1. Jak widać, nie jest to imponujący zestaw, niemniej jest to podstawowa klasa operująca na strumieniu wejściowym.

Tabela 6.1. Metody udostępniane przez klasę *InputStream*

Deklaracja metody	Opis
<code>int available()</code>	Zwraca przewidywaną liczbę bajtów, które mogą być pobrane ze strumienia przy najbliższym odczycie.
<code>void close()</code>	Zamyka strumień i zwalnia związane z nim zasoby.
<code>void mark(int readlimit)</code>	Zaznacza bieżącą pozycję w strumieniu.
<code>boolean markSupported()</code>	Sprawdza, czy strumień może obsługiwać metody <code>mark</code> i <code>reset</code> .
<code>abstract int read()</code>	Odczytuje kolejny bajt ze strumienia.
<code>int read(byte[] b)</code>	Odczytuje ze strumienia liczbę bajtów nie większą niż rozmiar tablicy <code>b</code> . Zwraca rzeczywiście odczytaną liczbę bajtów.
<code>int read(byte[] b, int off, int len)</code>	Odczytuje ze strumienia liczbę bajtów nie większą niż określona przez <code>len</code> i zapisuje je w tablicy <code>b</code> , począwszy od komórki wskazywanej przez <code>off</code> . Zwraca rzeczywiście przeczytaną liczbę bajtów.
<code>void reset()</code>	Wraca do pozycji strumienia wskazywanej przez wywołanie metody <code>mark</code> .
<code>long skip(long n)</code>	Pomija w strumieniu liczbę bajtów wskazywanych przez <code>n</code> . Zwraca rzeczywiście pominiętą liczbę bajtów.

Widać wyraźnie, że odczyt bajtów ze strumienia można przeprowadzić za pomocą jednej z metod o nazwie `read`. Przyjrzyjmy się metodzie odczytującej tylko jeden bajt. Mimo że odczytuje ona bajt, zwraca wartość typu `int`. Jest tak dlatego, że zwracana wartość zawsze zawiera się w przedziale 0 – 255 (to tyle, ile może przyjmować jeden bajt). Tymczasem zmienna typu `byte` reprezentuje wartości „ze znakiem” w zakresie od –128 do +127¹. Spróbujmy zatem napisać prosty program, który odczyta wprowadzony z klawiatury znak, a następnie wyświetli ten znak oraz jego kod ASCII na ekranie. Kod realizujący przedstawione zadanie jest widoczny na listingu 6.1.

Listing 6.1.

```
import java.io.*;

public class Main {
    public static void main(String args[]) {
        System.out.print("Wprowadź dowolny znak z klawiatury: ");
        try{
            char c = (char) System.in.read();
            System.out.print("Wprowadzony znak to: ");
            System.out.println(c);
            System.out.print("Jego kod to: ");
            System.out.println((int) c);
        }
        catch(IOException e){
            System.out.println("Błąd podczas odczytu strumienia.");
        }
    }
}
```

¹ Oczywiście reprezentowanie zakresu od 0 do 255 byłoby możliwe również za pomocą typów `short` i `long`.

Pierwszym krokiem jest zaimportowanie pakietu `java.io` (lekcja 17.), znajduje się w nim bowiem definicja wyjątku `IOException`, który musimy przechwycić. W metodzie `main` za pomocą metody `System.out.println` wyświetlamy tekst z prośbą o wprowadzenie dowolnego znaku z klawiatury, a następnie wykonujemy instrukcję:

```
char c = (char) System.in.read();
```

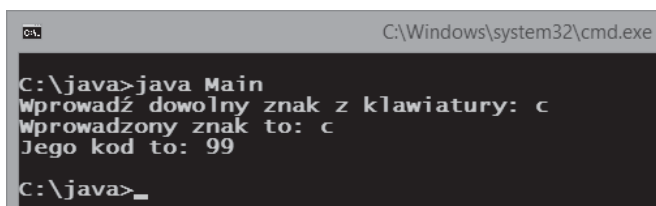
Wywołujemy zatem metodę `read` obiektu `in` zawartego w klasie `System`². Obiekt ten jest klasy `InputStream` (typem obiektu jest `InputStream`) i reprezentuje standardowy strumień wejściowy. W typowym przypadku jest on powiązany z klawiaturą, więc z tego strumienia będą odczytywane dane wprowadzane przez użytkownika z klawiatury. Metoda `read` zwraca wartość typu `int`, dokonujemy zatem konwersji na typ `char`, tak aby wyświetlić na ekranie wprowadzony znak, i przypisujemy go zmiennej `c`. Następnie wyświetlamy zawartość tej zmiennej za pomocą instrukcji:

```
System.out.println(c);
```

Aby wyświetlić kod znaku, należy ponownie dokonać konwersji. Zmienną `c` typu `char` konwertujemy na typ `int`, tak aby metoda `println` potraktowała ją jako liczbę, a nie jako znak. Ponieważ metoda `read` może spowodować powstanie wyjątku `IOException`, wszystkie instrukcje zostały ujęte w blok `try`, który zapobiegnie niekontrolowanemu zakończeniu programu (por. lekcje z rozdziału 4.). Przykładowy efekt działania programu z listingu 6.1 jest widoczny na rysunku 6.1.

Rysunek 6.1.

Przykładowy efekt działania programu z listingu 6.1



```
C:\Windows\system32\cmd.exe
C:\java>java Main
Wprowadź dowolny znak z klawiatury: c
Wprowadzony znak to: c
Jego kod to: 99
C:\java>_
```

Trzeba zwrócić uwagę, że w ten sposób tak naprawdę odczytywany jest ze strumienia nie jeden znak, ale jeden bajt. A te pojęcia nie są tożsame. Jeden znak może zawierać w zależności od standardu kodowania od jednego do kilku bajtów.

Wczytywanie tekstu

Wiadomo już, jak odczytać jeden bajt, co jednak zrobić, aby wprowadzić całą linię tekstu? Przecież taka sytuacja jest o wiele częstsza. Można oczywiście odczytywać pojedyncze znaki w pętli tak długo, aż zostanie osiągnięty znak końca linii, oraz połączyć je w obiekt typu `String`. Najlepiej byłoby wręcz wyprowadzić z klasy `InputStream` klasę pochodną, która zawierałaby metodę np. o nazwie `readLine`, wykonującą taką czynność. Trzeba byłoby przy tym pamiętać o odpowiedniej obsłudze standardów kodowania znaków. Na szczęście w JDK został zdefiniowany cały zestaw klas operujących na strumieniach wejściowych. Zamiast więc powtarzać coś, co zostało już zrobione, najlepiej po prostu z nich skorzystać.

² Obiekt `in` jest polem finalnym i statycznym klasy `System`.

Zadanie to będzie wymagało użycia dwóch klas pośredniczących. Klasą udostępniającą metodę `readLine`, która jest w stanie prawidłowo zinterpretować znaki przychodzące ze strumienia, jest `BufferedReader`. Aby jednak można było utworzyć obiekt takiej klasy, musi powstać obiekt klasy `Reader` lub klasy od niej pochodnej — w tym przypadku najodpowiedniejszy będzie `InputStreamReader`. Ponieważ nie ma takiego obiektu w aplikacji (do dyspozycji mamy jedynie obiekt `System.in` typu `InputStream`), należy go utworzyć, wywołując odpowiedni konstruktor. Będzie to obiekt (strumień) pośredniczący w wymianie danych. Do jego utworzenia potrzeba z kolei obiektu klasy `InputStream`, a tym przecież dysponujemy. Dlatego też kod programu, który odczytuje linię tekstu ze standardowego wejścia, będzie wyglądać tak, jak zostało to przedstawione na listingu 6.2.

Listing 6.2.

```
import java.io.*;

public class Main {
    public static void main(String args[]) {
        BufferedReader brIn = new BufferedReader(
            new InputStreamReader(System.in)
        );
        System.out.println("Wprowadź linię tekstu zakończoną znakiem Enter:");
        try{
            String line = brIn.readLine();
            System.out.print("Wprowadzona linia to: " + line);
        }
        catch(IOException e){
            System.out.println("Błąd podczas odczytu strumienia.");
        }
    }
}
```

Pierwszym zadaniem jest utworzenie obiektu `brIn` klasy `BufferedReader`. Mamy tu do czynienia ze złożoną instrukcją, która najpierw tworzy obiekt typu `InputStreamReader`, wykorzystując obiekt `System.in`, i dopiero ten obiekt przekazuje konstruktorowi klasy `BufferedReader`. Zatem konstrukcję:

```
BufferedReader brIn = new BufferedReader(
    new InputStreamReader(System.in)
);
```

można również rozbić na dwie części:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader brIn = new BufferedReader(isr);
```

Znaczenie będzie takie samo, jednak w drugim przypadku zostanie utworzona zupełnie niepotrzebnie dodatkowa zmienna `isr` typu `InputStreamReader`. To, który ze sposobów jest czytelniejszy i zostanie zastosowany, zależy jednak od indywidualnych preferencji programisty. Z reguły stosuje się sposób pierwszy.

Kiedy obiekt klasy `BufferedReader` jest już utworzony, można wykorzystać metodę `readLine`, która zwróci w postaci obiektu typu `String` (klasa `String` reprezentuje ciągi

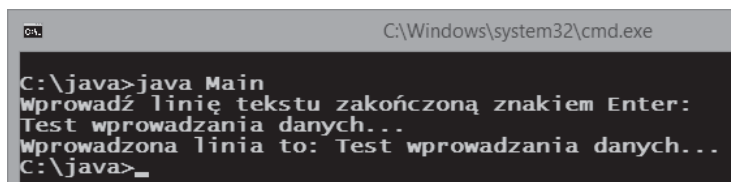
znaków) linię tekstu wprowadzoną przez użytkownika. Linia tekstu jest rozumiana jako ciąg znaków wprowadzony aż do naciśnięcia klawisza *Enter* (dokładniej: aż do osiągnięcia znaku końca wiersza w strumieniu). Wynika z tego, że instrukcja:

```
String line = brIn.readLine();
```

utworzy zmienną *line* typu *String* i przypisze jej obiekt (referencję do obiektu) zwrócony przez metodę *readLine* obiektu *brIn*. Działanie programu będzie zatem następujące: po uruchomieniu zostanie wyświetlona prośba o wprowadzenie linii tekstu, następnie linia ta zostanie odczytana i przypisana zmiennej *line*, a potem zawartość tej zmiennej zostanie wyświetlona na ekranie. Przykład wykonania tego programu pokazano na rysunku 6.2.

Rysunek 6.2.

*Wczytanie linii tekstu
za pomocą obiektu
klasy *BufferedReader**



```
C:\Windows\system32\cmd.exe
C:\java>java Main
Wprowadź linię tekstu zakończoną znakiem Enter:
Test wprowadzania danych...
Wprowadzona linia to: Test wprowadzania danych...
C:\java>_
```

Skoro wiadomo już, jak wczytać wiersz tekstu z klawiatury, spróbujmy napisać program, którego zadaniem będzie wczytywanie kolejnych linii ze standardowego wejścia tak długo, aż zostanie odczytany ciąg znaków *quit*. Zadanie wydaje się banalne, wystarczy przecież do kodu z listingu 6.2 dodać pętlę *while*, która będzie sprawdzała, czy zmienna *line* zawiera napis *quit*. W pierwszym odruchu napiszemy zapewne pętlę, która będzie wyglądać następująco (przy założeniu, że wcześniej został prawidłowo zainicjowany obiekt *brIn*):

```
String line = "";
while(line != "quit"){
    line = brIn.readLine();
    System.out.println("Wprowadzona linia to: " + line);
}
System.out.println("Koniec wczytywania danych.");
```

Warto przeanalizować ten przykład. Na początku deklarujemy zmienną *line* i przypisujemy jej pusty ciąg znaków. W pętli *while* sprawdzamy warunek, czy zmienna *line* jest różna od ciągu znaków *quit* — jeśli tak, to pętla jest kontynuowana, a jeśli nie, to kończy działanie. Zatem znaczenie jest następujące: dopóki *line* jest różne od *quit*, wykonuj instrukcje z wnętrza pętli. Wewnątrz pętli przypisujemy natomiast zmiennej *line* wiersz tekstu odczytany ze standardowego strumienia wejściowego oraz wyświetlamy go na ekranie.

Wszystko to wygląda bardzo poprawnie, ma tylko jedną wadę — nie zadziała zgodnie z założeniami. Aby się o tym przekonać, wystarczy uruchomić program z listingu 6.3, następnie wpisać kilka linii tekstu, a po nich ciąg znaków *quit*. Teoretycznie program powinien zakończyć działanie, tymczasem działa nadal, co jest widoczne na rysunku 6.3.

Rysunek 6.3.
 Błąd w pętli `while`
 uniemożliwia
 zakończenie programu

```

C:\Windows\system32\cmd.exe - java Main
C:\java>java Main
Wprowadzaj linie tekstu. Aby zakończyć, wpisz quit.
test
Wprowadzona linia to: test
quit
Wprowadzona linia to: quit
quit
Wprowadzona linia to: quit

```

Listing 6.3.

```

import java.io.*;

public class Main {
    public static void main(String args[]) {
        BufferedReader brIn = new BufferedReader(
            new InputStreamReader(System.in)
        );
        System.out.println("Wprowadzaj linie tekstu. Aby zakończyć, wpisz quit.");
        String line = "";
        try{
            while(line != "quit"){
                line = brIn.readLine();
                System.out.println("Wprowadzona linia to: " + line);
            }
            System.out.println("Koniec wczytywania danych.");
        }
        catch(IOException e){
            System.out.println("Błąd podczas odczytu strumienia.");
        }
    }
}

```

Co się dzieje, gdzie jest błąd? Podejrzanie powinno najpierw paść na warunek zakończenia pętli `while`. On rzeczywiście jest nieprawidłowy, choć wydaje się, że instrukcja ta jest poprawna. Zwróćmy jednak uwagę na to, co tak naprawdę jest w tym warunku porównywane. Miał być porównany ciąg znaków zapisany w zmiennej `line` z ciągiem znaków `quit`. Tymczasem po umieszczeniu w kodzie sekwencji `"quit"` powstał nieznanymy obiekt klasy `String` faktycznie zawierający ciąg znaków `quit`, a w miejsce napisu `"quit"` została wstawiona referencja do tego obiektu. Jak wiadomo z wcześniejszych lekcji (lekcja 13.), zmienna `line` jest także referencją do obiektu klasy `String`. Instrukcja porównania `line != "quit"` porównuje zatem ze sobą dwie REFERENCJE, które w tym przypadku muszą być różne. W związku z tym nie dochodzi tu do porównania zawartości obiektów. Dlatego program nie może działać poprawnie i wpada w nieskończoną pętlę (aby zakończyć jego działanie, trzeba wcisnąć na klawiaturze kombinację klawiszy `Ctrl+C`).

Koniecznym jest więc zapamiętać, że do porównywania obiektów nie używa się operatorów porównywania! Zamiast tego należy skorzystać z metody `equals`. Jest ona zadeklarowana w klasie `Object` i w związku z tym dziedziczona przez wszystkie klasy.

Warto pamiętać o tym podczas tworzenia własnych klas, by w razie potrzeby dopisać własną wersję tej metody.

Jak zastosować tę metodę w praktyce? Otóż zwraca ona wartość `true`, kiedy obiekty są takie same (czyli ich zawartość jest taka sama), lub `false`, kiedy obiekty są różne. W przypadku obiektów typu `String` wartość `true` zostanie zwrócona, jeśli ciąg znaków zapisany w jednym z nich jest taki sam jak ciąg znaków zapisany w drugim. Jeżeli np. istnieje zmienna zadeklarowana jako:

```
String napis1 = "test";
```

to wynikiem operacji:

```
napis1.equals("test");
```

będzie wartość `true`, a wynikiem operacji:

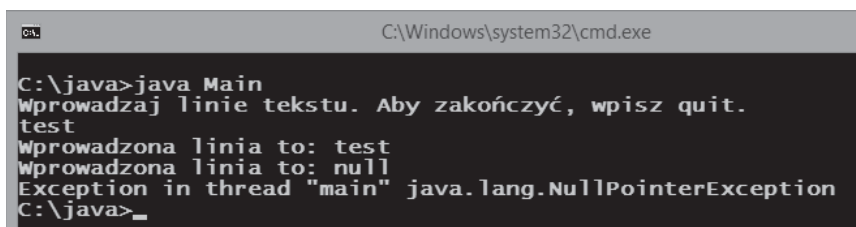
```
napis1.equals("java");
```

będzie wartość `false`.

Powróćmy teraz do pętli `while`. Jako że znane są nam już właściwości metody `equals`, w pierwszej chwili na pewno na myśl przyjdzie nam warunek:

```
while(!line.equals("quit")){  
    /*instrukcje pętli while*/  
}
```

Jeśli wprowadzimy go do kodu z listingu 6.3, aplikacja zacznie poprawnie reagować na polecenie `quit` — będzie się wydawać, że całość nareszcie działa prawidłowo. W zasadzie można byłoby się zgodzić z tym stwierdzeniem, gdyż program rzeczywiście działa zgodnie z założeniami, tylko że teraz zawiera kolejny błąd, tym razem dużo trudniejszy do wykrycia. Ujawni się on dopiero przy próbie przerwania pracy aplikacji. Spróbujmy uruchomić program, a po wpisaniu testowej linii tekstu przerwijmy jego działanie (w większości systemów należy wcisnąć kombinację klawiszy `Ctrl+C`). Efekt jest widoczny na rysunku 6.4. Zapewne nie spodziewaliśmy się wyjątku `NullPointerException`...



```
C:\Windows\system32\cmd.exe  
C:\java>java Main  
Wprowadzaj linie tekstu. Aby zakończyć, wpisz quit.  
test  
Wprowadzona linia to: test  
Wprowadzona linia to: null  
Exception in thread "main" java.lang.NullPointerException  
C:\java>_
```

Rysunek 6.4. Ukryty błąd w programie spowodował wygenerowanie wyjątku

Na przyczynę powstania tego błędu naprowadza linia tekstu poprzedzająca komunikat maszyny wirtualnej, mianowicie: `Wprowadzona linia to: null`. Oznacza to bowiem (spójrzmy na kod), że metoda `readLine` obiektu `brIn` zwróciła — zamiast ciągu znaków — wartość `null`. Jest to standardowe działanie, metoda ta zwraca wartość `null`, kiedy zostanie osiągnięty koniec strumienia, czyli kiedy nie ma w nim już żadnych danych do odczytania. Co się dzieje dalej? Otóż wartość `null` jest przypisywana zmiennej `line`,

a potem w warunku pętli `while` następuje próba wywołania metody `equals` obiektu wskazywanego przez `line`. Tymczasem `line` ma wartość `null` i nie wskazuje na żaden obiekt. W takiej sytuacji musi zostać wygenerowany wyjątek `NullPointerException`.

Jak temu zapobiec? Możliwości są dwie: albo dodamy w pętli warunek sprawdzający, czy `line` jest równe `null`, a jeśli tak, to przerwiemy pętlę np. instrukcją `break`, albo też zmodyfikujemy sam warunek pętli. Ta druga możliwość jest lepsza, gdyż nie powoduje wykonywania dodatkowego kodu, a jednocześnie otrzymujemy ciekawą konstrukcję. Poprawiony warunek powinien wyglądać następująco:

```
while(!"quit".equals(line)){
    /*instrukcje pętli while*/
}
```

Początkowo tego typu zapis budzi zdziwienie: jak można wywoływać jakąkolwiek metodę na rzecz ciągu znaków? Wystarczy jednak sobie przypomnieć stwierdzenie, które pojawiło się kilka akapitów wyżej, otóż literał³ `"quit"` w rzeczywistości powoduje powstanie obiektu klasy `String` i podstawianie w jego (literału) miejsce referencji do tego obiektu. Skoro tak, można wywołać metodę `equals`. To właśnie dzięki takiej konstrukcji unikamy wyjątku `NullPointerException`, gdyż nawet jeśli `line` będzie miało wartość `null`, to metoda `equals` po prostu zwróci wartość `false`, bo `null` na pewno jest różne od ciągu znaków `quit`. Ostatecznie pełny prawidłowy kod będzie miał postać przedstawioną na listingu 6.4. Przykład działania tego kodu został z kolei zaprezentowany na rysunku 6.5.

Listing 6.4.

```
import java.io.*;

public class Main {
    public static void main(String args[]) {
        BufferedReader brIn = new BufferedReader(
            new InputStreamReader(System.in)
        );
        System.out.println("Wprowadzaj linie tekstu. Aby zakończyć, wpisz quit.");
        String line = "";
        try{
            while(!"quit".equals(line)){
                line = brIn.readLine();
                System.out.println("Wprowadzona linia to: " + line);
            }
            System.out.println("Koniec wczytywania danych.");
        }
        catch(IOException e){
            System.out.println("Błąd podczas odczytu strumienia.");
        }
    }
}
```

³ Czyli inaczej stała znakowa (napisowa), stały ciąg znaków umieszczony w kodzie programu.

Rysunek 6.5.

Wprowadzanie w pętli kolejnych linii tekstu

```

C:\Windows\system32\cmd.exe
C:\java>java Main
Wprowadzaj linie tekstu. Aby zakończyć, wpisz quit.
Pierwszy test
Wprowadzona linia to: Pierwszy test
Drugi test
Wprowadzona linia to: Drugi test
quit
Wprowadzona linia to: quit
Koniec wczytywania danych.
C:\java>_

```

Wprowadzanie liczb

Potrąfimy już odczytywać w aplikacji wiersze tekstu wprowadzane z klawiatury, równie ważną umiejętnością jest wprowadzanie liczb. Jak to zrobić? Trzeba sobie uzmysłowić, że z klawiatury zawsze wprowadzany jest tekst. Jeśli próbujemy wprowadzić do aplikacji wartość 123, to w rzeczywistości wprowadzimy trzy znaki, "1", "2" i "3" o kodach ASCII 61, 62, 63. Mogą one zostać przedstawione w postaci ciągu "123", ale to dopiero aplikacja musi przetworzyć ten ciąg na wartość 123. Takiej konwersji w przypadku wartości całkowitej można dokonać np. za pomocą metody `parseInt` z klasy `Integer`. Jest to metoda statyczna, można ją więc wywołać, nie tworząc obiektu klasy `Integer`⁴. Przykładowe wywołanie może wyglądać następująco:

```
int liczba = Integer.parseInt(ciąg_znaków);
```

Zmiennej `liczba` zostanie przypisana wartość typu `int` zawarta w ciągu znaków `ciąg_znaków`. Gdyby ciąg przekazany jako argument metody `parseInt` nie zawierał poprawnej wartości całkowitej, zostałby wygenerowany wyjątek `NumberFormatException`. Aby zatem wprowadzić do aplikacji wartość całkowitą, można odczytać wiersz tekstu, korzystając z metody `readLine`, a następnie wywołać metodę `parseInt`. Ten właśnie sposób został wykorzystany w programie z listingu 6.5. Zadaniem tego programu jest wczytanie liczby całkowitej oraz wyświetlenie wyniku mnożenia tej liczby przez wartość 2.

Listing 6.5.

```

import java.io.*;

public class Main {
    public static void main(String args[]) {
        BufferedReader brIn = new BufferedReader(
            new InputStreamReader(System.in)
        );

        System.out.print("Wprowadź liczbę całkowitą: ");

        String line = null;
        try{
            line = brIn.readLine();
        }
    }
}

```

⁴ Jak widać, metody statyczne, które były przedmiotem lekcji 18., przydają się w praktyce.

```

catch(IOException e){
    System.out.println("Błąd podczas odczytu strumienia.");
    return;
}

int liczba;

try{
    liczba = Integer.parseInt(line);
}
catch(NumberFormatException e){
    System.out.print("Podana wartość nie jest liczbą całkowitą.");
    return;
}
System.out.print(liczba + " * 2 = " + liczba * 2);
}
}

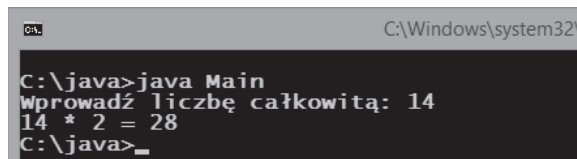
```

Kod zaczyna się od utworzenia obiektu `brIn` klasy `BufferedReader` powiązanej poprzez obiekt pośredniczący klasy `InputStreamReader` ze standardowym strumieniem wejściowym. Stosujemy technikę opisaną przy omawianiu przykładu z listingu 6.2. Następnie wyświetlamy prośbę o wprowadzenie liczby całkowitej oraz odczytujemy wiersz tekstu za pomocą metody `readLine` obiektu `brIn`, nie zapominając o przechwyceniu ewentualnego wyjątku `IOException`. W przypadku wystąpienia takiego wyjątku wyświetlamy stosowną informację na ekranie oraz kończymy wykonywanie programu. Warto zwrócić uwagę, że używana jest w tym celu instrukcja `return` bez żadnych parametrów, co oznacza wyjście z funkcji `main` (bez zwracania wartości), a tym samym zakończenie działania aplikacji.

Jeśli odczyt wiersza tekstu się powiedzie, zostanie on zapisany w zmiennej `line`. Deklarujemy więc dalej zmienną `liczba` typu `int` oraz przypisujemy jej wynik działania metody `parseInt` z klasy `Integer`. Metodzie przekazujemy ciąg znaków zapisany w zmiennej `line`. Jeśli wprowadzony przez użytkownika ciąg znaków nie reprezentuje poprawnej wartości liczbowej, wygenerowany zostaje wyjątek `NumberFormatException`. W takim wypadku wyświetlamy komunikat o tym fakcie oraz kończymy działanie funkcji `main`, a tym samym programu, wywołując instrukcję `return`. Jeśli jednak konwersja tekstu na liczbę powiedzie się, odpowiednia wartość zostanie zapisana w zmiennej `liczba`, można zatem wykonać mnożenie `liczba * 2` i wyświetlić wartość wynikającą z tego mnożenia na ekranie. Przykładowy wynik działania programu jest widoczny na rysunku 6.6.

Rysunek 6.6.

Przykładowy wynik działania programu z listingu 6.5



```

C:\Windows\system32\
C:\java>java Main
Wprowadź liczbę całkowitą: 14
14 * 2 = 28
C:\java>_

```

Gdyby miała być wczytana liczba zmiennoprzecinkowa, należałoby do konwersji zastosować metodę `parseDouble` z klasy `Double` lub `parseFloat` z klasy `Float`, co jest doskonałym ćwiczeniem do samodzielnego wykonania.

Skorowidz

A

- adapter MouseInputAdapter, 390
- aplety, 351, 354
 - konstrukcja, 355
 - parametry, 356
 - tworzenie, 352
- aplikacja, 351, 386
 - Eclipse, 6
 - jEdit, 6
 - NetBeans, 6
 - Notepad++, 6
- argumenty
 - finalne, 162
 - konstruktorów, 111
 - metod, 100
- ASCII, 270

B

- blok
 - default, 56
 - else, 48
 - finally, 192
 - switch, 55
 - try...catch, 165, 169–171, 180
- błąd
 - arytmetyczny, 179
 - implementacji interfejsu, 234
 - kompilacji, 31, 39, 127, 143, 197, 256
 - w pętli while, 274
 - w programie, 275
- bufor, 315

C

- ciąg
 - formatujący, 289
 - znaków quit, 330
- czcionki, 359

D

- deklaracja, 24
 - metody, 120
 - tablicy, 77, 80, 85
 - typu uogólnionego, 345
 - zmiennej typu klasowego, 93
- deklaracje
 - proste, 24
 - wielu zmiennych, 25
- dekrementacja, 35
- domyślna strona kodowa, 293
- domyślne wartości typów danych, 99
- dookreślanie typu, 341
- dostęp
 - chroniony, protected, 133
 - do katalogu, 298
 - do klasy wewnętrznej, 253
 - do klasy zewnętrznej, 245
 - do obiektu, 248
 - do składowych, 261
 - prywatny, private, 131
 - publiczny, public, 130
- dynamiczna zmiana pojemności, 327
- dynamiczny wektor elementów, 328
- działania arytmetyczne, 34
- działanie
 - etykietowanej instrukcji break, 72
 - etykietowanej instrukcji continue, 70, 72
 - konstruktorów odziedziczonych, 126
 - operatora dekrementacji, 37
 - operatora inkrementacji, 36
 - pętli for, 77
 - przeciążonej metody, 203
- dziedziczenie, 121, 122
 - interfejsów, 235, 238
 - po klasie zewnętrznej, 252
- dzielenie przez zero, 173
- dźwięki, 383

E

edytor tekstowy, 6
 elementy tablic, 76
 elipsa, 369
 enkapsulacja, 134
 etykiety, 71
 puste, 405
 tekstowe, 404, 405

F

figury geometryczne, 366
 finalne
 argumenty metod, 162
 klasy, 156
 metody, 161
 pola, 157, 159
 formatowanie, 290

G

grafika, 366

H

hermetyzacja, 134
 hierarchia
 klas, 224
 wyjątków, 176

I

iloczyn logiczny, 43
 implementacja
 interfejsu, 223
 interfejsu MouseListener, 401
 wielu interfejsów, 230
 importowanie pakietu, 271
 inferowanie typu, 341
 informacje
 o kliknięciach, 379
 o stanie załadowania obrazu, 375
 inicjalizacja
 obiektu, 110
 pól finalnych, 160
 tablicy, 80, 85
 inkrementacja, 35
 instalacja
 JDK, 9
 w systemie Linux, 11
 w systemie Windows, 10
 instrukcja, 23
 break, 55, 66–68
 continue, 68–71
 if...else, 47

if...else if, 51
 Math.sqrt(), 50
 return, 97
 switch, 53–57
 System.out.print, 33, 87
 System.out.println, 28
 throw, 183, 190
 instrukcje sterujące, 47
 interfejs, 222, 224
 ActionListener, 393, 397
 AudioClip, 383
 DocumentListener, 408
 Drawable, 224, 228, 229
 Iterable, 333–335
 Iterator, 335
 jako typ zwracany, 254
 MouseListener, 376–380, 385, 390, 401
 MouseMotionListener, 380
 WindowListener, 387
 interfejsy funkcjonalne, 261, 264

J

jawna konwersja typu, 197
 JDK, Java Development Kit, 6
 JRE, Java Runtime Environment, 6, 10

K

katalogi, 298, 304
 pobranie zawartości, 298
 tworzenie, 302
 usuwanie, 304
 klasa, 17, 91
 ActionEvent, 394, 398
 Applet, 365
 Applet, 352, 383
 ArrayList, 328
 BufferedInputStream, 321
 BufferedOutputStream, 321
 BufferedReader, 272, 320
 CheckboxMenu, 398
 Color, 363
 Component, 403
 Console, 288, 289
 Dane, 118
 DivideByZeroException, 190
 Double, 278
 Exception, 183
 File, 295, 302
 FileInputStream, 316
 FileOutputStream, 318
 FileReader, 316, 318
 FileWriter, 318, 411
 Font, 359
 FontMetrics, 361

- Frame, 386
- GeneralException, 189
- Graphics, 362, 366
- GraphicsEnvironment, 360
- ImageObserver, 375
- InputStream, 270, 271, 272
- Integer, 277
- JApplet, 352, 374
- JButton, 405
- JCheckBox, 411
- JCheckBoxMenuItem, 398, 399
- JComboBox, 413, 415
- JFrame, 386
- JLabel, 404
- JMenu, 391, 396
- JMenuBar, 391
- JMenuItem, 391, 392
- JPopupMenu, 400
- JTextArea, 410
- JTextComponent, 407
- JTextField, 407, 409
- Main, 94
- Matcher, 301
- MouseEvent, 377, 382
- MouseInputAdapter, 390
- MouseMotionAdapter, 390
- MyFilenameFilter, 300, 301
- MyMouseListener, 380
- MyWindowAdapter, 389
- Object, 201
- OutputStreamWriter, 293
- Pattern, 301
- Polygon, 372
- PrintStream, 291
- PrintWriter, 293, 294
- Punkt, 93, 103
- Punkt3D, 124
- RandomAccessFile, 307–312
- Reader, 272
- RuntimeException, 176, 177
- Scanner, 286–288
- Shape, 208, 213
- Stack, 331
- StreamTokenizer, 279–286
- SwingUtilities, 387
- Tablica, 167
- uruchomieniowa, 108
- WindowAdapter, 389
- klasy
 - abstrakcyjne, 212, 223
 - anonimowe, 257, 264
 - bazowe, 122
 - chronione, 251
 - finalne, 156
 - kontenerowe, 328
 - pakietowe, 141, 249, 250
 - pochodne, 225
 - potomne, 122
 - prywatne, 251
 - publiczne, 141, 250
 - statyczne, 260
 - statyczne zagnieżdżone, 239, 260
 - wewnętrzne, 141, 238, 240, 246, 249
 - wewnętrzne lokalne, 244
 - zagnieżdżone, 254
 - zewnętrzne, 241, 245
- klasyfikacja klas, 250
- kod
 - ASCII, 270
 - pośredni, 15
 - Unicode, 293
- kodowanie, 292
- kolory, 362
- koło, 369
- komenda
 - java, 13
 - javac, 13
- komentarz, 19
 - blokowy, 20
 - liniowy, 21
- kompilacja, 14, 15
- kompilacja wyrażenia regularnego, 301
- kompilator, 6, 15
- kompilator javac, 7, 14
- komponenty, 402
- komunikat o błędzie, 175, 237
- konflikt nazw, 232, 237, 238
- koniec pliku, 314
- konsola, 288
- konsola systemowa, 12
- konstrukcja apletu, 355
- konstruktor, 110, 212
 - argumenty, 111
 - bezargumentowy, 111
 - domyślny, 93, 218
 - klasy bazowej, 125
 - klasy potomnej, 125
 - przeciążanie, 112
 - wywoływanie metod, 115, 219
- konstruktory klasy
 - JCheckBox, 412
 - ComboBox, 413
 - JTextArea, 410
 - JTextField, 407
 - OutputStreamWriter, 293
- kontenery, 323, 339
- kontrola typów, 330, 337, 348
- konwersja typów prostych, 195, 196, 199
- kopiowanie plików, 312, 314, 320
- kreje pisma, 358, 359

L

liczba, 277
 linie, 368
 lista czcionek, 359, 361
 listy rozwijane, 413

M

maszyna wirtualna, 6
 menu, 391
 kaskadowe, 396, 397
 kontekstowe, 400, 401
 umożliwiające zaznaczanie, 399

metoda

actionPerformed, 393, 408
 actionPreformed, 395
 add, 330, 392
 addActionListener, 379
 addMouseListener, 379, 385
 addWindowListener, 387–390
 clearRect, 352
 createNewFile, 302
 delete, 304
 dispose, 390, 395
 draw, 213
 drawImage, 373
 drawOval, 369
 drawPolygon, 371
 drawRectangle, 370
 drawShape, 215
 drawString, 386
 exists, 303, 310
 fillOval, 369
 fillPolygon, 371
 fillRectangle, 370
 finalize, 118
 get, 326, 343
 getActionCommand, 395
 getAllFonts, 360
 getAudioClip, 384
 getAvailableFontFamilyNames, 360
 getButton, 377
 getCodeBase, 372
 getDocumentBase, 372
 getFontMetrics, 362
 getHeight, 362
 getImage, 372, 373
 getItem, 397
 getLocalGraphicsEnvironment, 360
 getMessage(), 174
 getParameter, 357
 getSource, 394, 399
 getValues, 347
 getX, 377
 getY, 377
 hasNext, 335

hasNextInt, 287
 imageUpdate, 374
 invokeLater, 387
 list, 298
 main, 106
 mkdir, 303
 makedirs, 303
 mouseClicked, 378, 380, 385
 mouseDragged, 380
 next, 335
 nextLine, 287
 nextToken, 280–282
 paint, 362
 parseDouble, 278
 parseInt, 277
 play, 385
 printf, 289
 read, 270, 271, 313
 readLine, 272, 275
 remove, 335
 resize, 325, 327
 save, 411
 set, 325, 326
 setActionCommand, 395
 setBounds, 405
 setColor, 362
 setDocumentListener, 408
 setFont, 362
 setJMenuBar, 392
 setLayout, 405
 setSize, 386
 setText, 405
 show, 345, 401
 size, 325
 start, 286
 stringWidth, 362
 super, 127
 System.gc, 118
 toString, 202
 windowClosing, 389
 write, 313

metody, 95
 argumenty, 100
 domyślne, 227, 229
 finalne, 161
 interfejsu DocumentListener, 409
 interfejsu Iterable, 334
 interfejsu MouseListener, 376
 interfejsu MouseMotionListener, 380
 interfejsu WindowListener, 388
 klasy anonimowej, 259
 klasy ArrayList, 329
 klasy Component, 403, 404
 klasy Console, 289
 klasy File, 296, 297, 298
 klasy FileInputStream, 316
 klasy FileOutputStream, 318

- klasy FileReader, 317
- klasy FileWriter, 318
- klasy Graphics, 366
- klasy InputStream, 270
- klasy JComboBox, 414
- klasy PrintStream, 291, 292
- klasy RandomAccessFile, 307, 308
- klasy Scanner, 286
- klasy Stack, 331
- prywatne, 210
- przeciążanie, 108, 144
- przesłanie, 146
- statyczne, 152, 277
- wywołanie polimorficzne, 208, 220, 259
- wywoływane w konstruktorach, 115, 219
- zwracanie wyniku, 97

mnożenie liczb, 284

model RGB, 363

mysz, 376

N

narzędzia, 6

narzędzia JDK, 11

nawias

- kątowny, 343
- klamrowy, 47, 95, 170
- kwadratowy, 75, 80, 82
- okrągły, 170, 196

nazwa

- klasy, 92
- pakietu, 138
- pliku, 92
- zmiennych, 26

negacja logiczna, 43

nieprawidłowa implementacja interfejsów, 234

niewłaściwy typ danych, 340, 344

niszczenie obiektu, 121

O

obiekt, 17, 94

- file, 298
- InputStreamReader, 272
- jako argument, 102
- klasy wewnętrznej, 246, 249
- klasy zewnętrznej, 249
- outp, 294
- strTok, 284
- System.in, 269, 272, 287

obsługa

- dźwięków, 383
- konsoli, 288
- menu, 391, 394
- myszy, 376
- okna, 390
- przycisków, 405
- zdarzeń, 389, 394, 397

odczyt danych z pliku, 309

odniesienie, 27

odśmiecacz, 120

odtwarzanie dźwięków, 383

odwołanie

- do nieistniejącego elementu, 166
- do prywatnego pola, 246
- do pustej zmiennej obiektowej, 179

odzyskiwanie pamięci, 118

okno wyboru składników JDK, 10

opcja encoding, 31, 293

operacje

- na plikach, 306
- na tablicach, 74
- na zmiennych, 33

operator

- +, 28
- dekrementacji, 35
- inkrementacji, 35
- kropki, 94
- new, 81, 85, 93
- warunkowy, 57

operatory

- arytmetyczne, 34
- bitowe, 41, 42
- logiczne, 42, 43
- porównywania, 44, 45
- przypisania, 44

P

pakiet, 137, 250

- java.awt, 352, 358
- java.io, 271
- java.util, 328
- javax.swing, 352, 402
- JDK, 6

parametr

- mode, 308
- this, 388

parametry

- apletu, 356
- zdarzenia, 381

pasek menu, 391

pętla, 59

- do...while, 63, 64
- for, 59–61
- foreach, 65, 333
- while, 62, 63, 273

pętle zagnieżdżone, 70, 82

pierwiastki równania kwadratowego, 284

plik, 304

- index.html, 354
- Main.class, 15
- Main.java, 14, 95

pliki

- dźwiękowe, 383
- kopiowanie, 312, 314, 320
- odczyt danych, 309, 316
- o dostępie swobodnym, 307
- operacje strumieniowe, 316
- tworzenie, 302
- usuwanie, 304
- zapis danych, 310, 318

pobranie

- tokena, 281
- zawartości katalogu, 298

pola, 91, 149

- finalne, 157
- finalne typów prostych, 158
- finalne typów referencyjnych, 159
- interfejsów, 226
- JCheckBox, 411
- niezainicjowane, 98
- statyczne, 153
- tekstowe, 407, 408

pole

- nval, 281
- sval, 281
- ttype, 280

polecenie

- chcp, 30
- cmd, 12
- import, 138
- javac, 14

polimorficzne wywoływanie metod, 208, 220, 259

polimorfizm, 195, 206

polskie znaki, 292

ponowne zgłaszanie wyjątków, 187

poprawność danych, 165

późne wiązanie, 204, 206

praklasa, 202

priorytety operatorów, 45

problem kontroli typów, 337

programowanie obiektowe, 91, 195

przechowywanie wielu danych, 323

przechwytywanie wielu wyjątków, 177

przeciążanie

- konstruktorów, 112
- metod, 108

przeglądanie kontenerów, 333

przekroczenie zakresu wartości, 41

przesłanianie

- metod, 144–149
- pól, 146, 149, 151

przyciski, 405

przypisanie, 24

punkty, 368

R

referencja, reference, 27

referencje do obiektu, 102, 114, 207

reprezentacja liczb, 42

RGB, 363

rodzaje

- klas wewnętrznych, 246, 249
- pól tekstowych, 407
- wyjątków, 173

rodziny czcionek, 361

rozmiar tablicy, 78, 326

równanie kwadratowe, 50, 284, 286

rysowanie

- elips, 369
- figur, 366
- kół, 369
- linii, 368
- prostokątów, 368, 370
- punktów, 368
- wielokątów, 370, 371

rzeczywisty typ obiektu, 204

rzutowanie

- na interfejs, 256
- na typ Object, 201
- na typy interfejsowe, 254
- na typy klasowe, 257
- obiektów, 195
- typów obiektowych, 128, 197, 201, 341
- w górę, 206

S

sekcja finally, 190

sekwencja ucieczki, 32

sekwencje znaków specjalnych, 32

składnia

- ExtendedOutside.Inside, 253
- Outside.Inside, 253

składowe

- klas wewnętrznych, 242
- klasy, 92
- RGB, 364
- statyczne, 152

skrypt startowy powłoki, 13

słowo kluczowe

- else, 47
- extends, 235
- final, 156
- implements, 226
- package, 137
- super, 211
- this, 114, 116
- void, 96, 100

specyfikator dostępu, 129
 private, 131
 protected, 133
 public, 130
 sprawdzanie poprawności danych, 165
 stała
 EXIT_ON_CLOSE, 387
 napisowa, 276
 stałe klasy
 Color, 363
 MouseEvent, 382
 stan zmiennej iteracyjnej, 60
 standard
 CP1250, 292
 CP852, 293
 Unicode, 30, 293
 standardowe
 wejście, 269
 wyjście, 279
 statyczność klasy wewnętrznej, 260
 sterta, heap, 94
 stos, stack, 94
 stosowanie uogólnień, 341
 strona kodowa, 293
 strona kodowa 1250, 30
 struktura
 programu, 13
 tablicy, 74
 tablicy dwuwymiarowej, 80
 strumieniowe operacje na plikach, 316
 strumień wejściowy, 269
 suma logiczna, 43
 symbol T, 343
 system plików, 295
 system wejścia-wyjścia, 269

Ś

ścieżka dostępu do katalogu, 298
 średnik, 258
 środowisko
 programistyczne, 6
 uruchomieniowe, 7, 10

T

tablice, 74
 dwuwymiarowe, 80
 dynamiczne, 326, 328
 nieregularne, 84, 89
 tekst, 271
 terminal, 12
 testowanie apletów, 354
 tokeny, 279, 280
 tworzenie
 apletu, 352
 aplikacji, 386

etykiet, 405
 interfejsów, 222
 katalogów, 302
 klas wewnętrznych, 239
 klasy anonimowej, 257–259
 list rozwijalnych, 413
 menu, 392
 nieregularnej tablicy dwuwymiarowej, 86
 obiektów, 105
 obiektów klas wewnętrznych, 248
 obiektu, 96, 98
 obiektu strTok, 281
 obiektu wyjątku, 183
 pakietów, 137
 paska menu, 394
 plików, 302
 tablicy, 80, 82
 wielu klas wewnętrznych, 240
 własnych wyjątków, 188
 tymczasowa zmienna referencyjna, 395
 typ

boolean, 19
 byte, 18
 char, 19
 double, 19
 float, 19
 int, 18
 long, 18
 Object, 201
 short, 18
 typy
 arytmetyczne całkowitoliczbowe, 18
 arytmetyczne zmiennopozycyjne, 19
 danych, 16
 generyczne, 336
 obiektowe, 26, 197
 proste, primitive types, 18
 uogólnione, 336

U

Unicode, 293
 uogólnianie metod, 345
 uogólnienia, 341
 uruchamianie apletu, 354
 usuwanie
 katalogów, 304
 komentarzy, 256
 plików, 304

W

wartości
 domyślne pól, 98
 parametru mode, 308
 pola ttype, 280

- wartość
 - aktualnego tokena, 280
 - null, 27, 120, 275
 - warunek zakończenia pętli, 274
 - wbudowane typy danych, 18
 - wczesne wiązanie, 208
 - wczytywanie
 - danych, 277, 283, 286, 288
 - grafiki, 372
 - tekstu, 271, 273, 277
 - wersje Javy, 7
 - wiązanie
 - czasu wykonania, 206
 - dynamiczne, 206
 - wielokąty, 370
 - wiersz
 - poleceń, 12
 - tekstu, 273
 - wirtualna maszyna Javy, 15
 - właściwość length, 78, 83
 - wprowadzanie, *Patrz* wczytywanie
 - współrzędne
 - biegunowe, 134
 - kliknięcia, 380
 - wybór składników JDK, 10
 - wyjątek, exception, 165, 169
 - ArithmeticException, 175, 179, 181
 - ArrayIndexOutOfBoundsException, 76, 165, 169, 324
 - ClassCastException, 205, 206, 337
 - DivideByZeroException, 189
 - FileNotFoundException, 308, 310
 - IOException, 271, 303
 - NullPointerException, 179, 181, 276
 - NumberFormatException, 277, 278, 365
 - PatternSyntaxException, 300
 - UnsupportedEncodingException, 293, 294
 - UnsupportedOperationException, 335
 - wyjątki własne, 182, 188
 - wykonanie programu, 15
 - wykorzystanie
 - kontenerów, 339
 - obiektów, 140
 - pakietów, 139
 - wyrażenia
 - lambda, 266
 - regularne, 299–302
 - wyświetlanie
 - elementów tablicy nieregularnej, 87
 - menu, 401
 - napisu, 386
 - polskich znaków, 294
 - przycisków, 405
 - systemowego komunikatu, 175
 - wartości zmiennych, 27
 - zawartości katalogu, 299
 - zawartości tablicy, 81
 - wywołania
 - konstruktorów, 216
 - metod, 115, 219
 - metod klas pochodnych, 204
 - metod klasy wewnętrznej, 247
 - metod nieistniejących, 225
 - metod przesłoniętych, 147
 - złożone konstruktorów, 219
- ## Z
- zagnieżdżanie bloków try...catch, 180, 186
 - zakończenie działania programu, 274
 - zakres
 - wartości zmiennej, 40
 - typów arytmetycznych, 18
 - typów zmiennoprzecinkowych, 19
 - zapis do pliku, 310, 318
 - zaznaczanie pozycji, 399
 - zdarzenia
 - myszki, 377
 - związane z oknem, 387
 - zgłaszanie
 - wyjątku, 182
 - ponowne wyjątku, 185
 - zmiana
 - czcionki, 31
 - strony kodowej, 30
 - zmienna środowiskowa
 - CLASSPATH, 139, 140
 - PATH, 13
 - zmiennie, 23, 102
 - referencyjne, 94
 - typów odnośnikowych, 26
 - znacznik
 - <applet>, 353
 - <object>, 353
 - znaczniki formatujące printf, 290
 - znak %, 289
 - znaki
 - polskie, 29
 - specjalne, 32
 - zwracanie
 - wielu wartości, 346
 - wyniku, 97

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>



PRAKTYCZNY KURS

Znajomość Javy, jako jednego z najpopularniejszych języków programowania na świecie, to dziś obowiązek każdego programisty. Na nic nie zdadzą się wykręty, że przecież można inaczej — pamiętaj, Java jest wszędzie! Dlatego jeśli masz ambicje zając się programowaniem albo szybko nauczyć się podstaw nowego języka, natychmiast musisz zaopatrzyć się w tę książkę. Pomoże Ci ona zrozumieć składnię i logikę Javy, a także opanować polecenia, byś mógł szybko rozpocząć samodzielną pracę w tym języku.

Z książki dowiesz się, jak zainstalować Javę i jakie narzędzia będą Ci potrzebne. Odkryjesz podstawowe zasady pisania programów z uwzględnieniem obiektowego charakteru tego języka, nauczysz się stosować instrukcje i odpowiednie typy danych, a także wykorzystywać deklaracje, klasy i metody. Kolejne rozdziały wskażą Ci, do czego służą pętle, tablice i operatory. Zanim się obejrzyś, będziesz już umiał dokonywać konwersji typów, używać konstruktorów i zagnieżdżać klasy, a stąd już tylko krok do tworzenia apletów i aplikacji. Przekonaj się, jak szybko możesz poznać Javę — wystarczy seria konkretnych ćwiczeń, a Twoje umiejętności posybią w górę!

- Krótka historia Javy, narzędzia i wersje
- Struktura programu, kompilacja i wykonanie
- Podstawy obiektowości, typy danych, komentarze, deklaracje i przypisania
- Wyprowadzanie danych na ekran i operacje na zmiennych
- Instrukcje if...else, break i continue, switch i operator warunkowy
- Pętle, podstawowe operacje na tablicach i tablice wielowymiarowe
- Klasy, pola, metody, argumenty i przeciążanie metod
- Konstruktory, klasy potomne, specyfikatory dostępu i pakiety
- Przesłanianie metod, składowe statyczne, klasy i składowe finalne, blok try...catch
- Konwersje typów, rzutowanie obiektów, klasy abstrakcyjne i tworzenie interfejsów
- Rodzaje klas wewnętrznych, dziedziczenie, klasy anonimowe i zagnieżdżone
- Standardowe wejście i wyjście, system plików, operacje na plikach
- Kontenery, typy uogólnione, podstawy apletów
- Kroje pisma (fonty) i kolory, grafika, dźwięki i obsługa myszy
- Tworzenie aplikacji i komponenty

Napisz swój pierwszy program w Javie!

sięgnij po **WIĘCEJ**

Helion

26437

numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

● <http://helion.pl/promocje>

Książki najchętniej czytane:

● <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

● <http://helion.pl/nowosci>

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



KOD KORZYŚCI

ISBN 978-83-246-9663-5



9 788324 696635

Informatyka w najlepszym wydaniu

cena: 69,00 zł