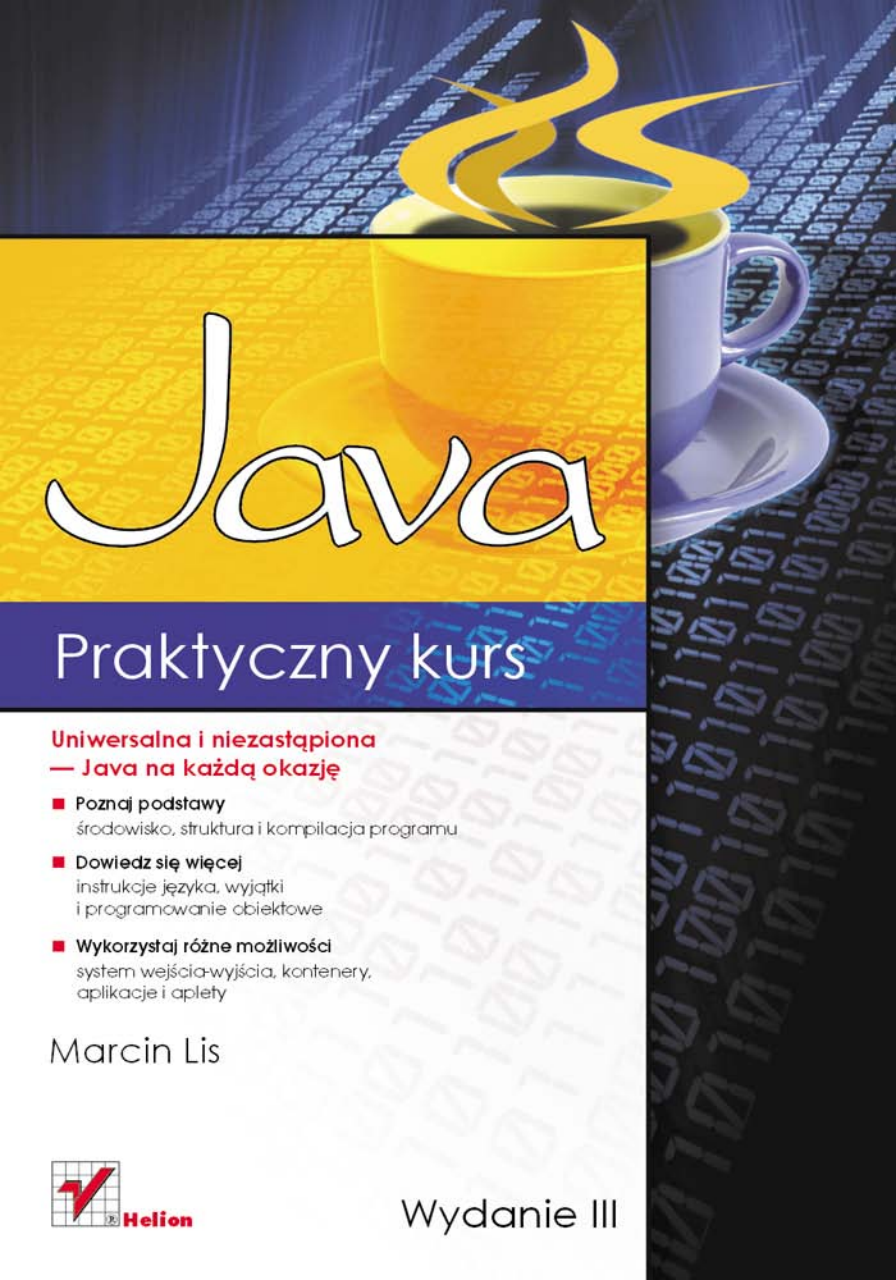




Java

Praktyczny kurs

**Uniwersalna i niezastąpiona
— Java na każdą okazję**

- **Poznaj podstawy**
środowisko, struktura i kompilacja programu
- **Dowiedz się więcej**
instrukcje języka, wyjątki
i programowanie obiektowe
- **Wykorzystaj różne możliwości**
system wejścia-wyjścia, kontenery,
aplikacje i applety

Marcin Lis



Wydanie III

» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» 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

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Praktyczny kurs Java. Wydanie III

Autor: [Marcin Lis](#)

ISBN: 978-83-246-3228-2

Format: 158×235, stron: 400



Uniwersalna i niezastąpiona – Java na każdą okazję

- Poznaj podstawy – środowisko, struktura i kompilacja programu
- Dowiedz się więcej – instrukcje języka, wyjątki i programowanie obiektowe
- Wykorzystaj różne możliwości – system wejścia-wyjścia, kontenery, aplikacje i applety

Język Java nieprzerwanie święci triumfy na salonach profesjonalnych firm, zajmujących się programowaniem. Jest wykorzystywany zarówno w prostych programach dla telefonów komórkowych, jak i w skomplikowanych aplikacjach sieciowych. Jego główne zalety to duża przenośność i świetna, przemyślana konstrukcja, która pozwala łatwo opanować zasady programowania i szybko zacząć tworzyć własne, dobrze działające programy. Java ma jeszcze jedną cechę, istotną z punktu widzenia każdej osoby zajmującej się informatyką – po prostu nie wypada jej nie znać!

Książka „Praktyczny kurs Java. Wydanie III” oferuje swoim czytelnikom możliwość łatwego i szybkiego zapoznania się z podstawami programowania w tym języku. Z jej pomocą w mig zainstalujesz odpowiednie środowisko programistyczne i poznasz reguły budowania aplikacji w Javie, typy danych oraz rodzaje zmiennych. Nauczysz się kontrolować przebieg wykonywania programu oraz wykorzystywać tablice. Zrozumiesz, na czym polega programowanie obiektowe i związane z nim podstawowe pojęcia, takie jak dziedziczenie i polimorfizm. Dowiesz się, jak obsługiwać i tworzyć wyjątki, jak działa system wejścia-wyjścia, co to są kontenery i typy uogólnione oraz czym różnią się aplikacje od appletów. A wszystko to w serii znakomitych, praktycznych ćwiczeń!

- Krótka historia Javy, jej narzędzia i wersje
- Instalacja JDK i podstawy programowania
- Zmienne, instrukcje sterujące i tablice
- Dziedziczenie, polimorfizm, interfejsy i klasy wewnętrzne
- Wyjątki
- System wejścia-wyjścia
- Kontenery i typy uogólnione
- Aplikacje i applety

Zanurz się w świecie Javy!

Spis treści

Wstęp	5
Rozdział 1. Podstawy	9
Instalacja JDK	9
Instalacja w systemie Windows	10
Instalacja w systemie Linux	10
Przygotowanie do pracy z JDK	11
Podstawy programowania	13
Lekcja 1. Struktura programu, kompilacja i wykonanie	13
Lekcja 2. Podstawy obiektowości i typy danych	15
Lekcja 3. Komentarze	18
Rozdział 2. Instrukcje języka	21
Zmienne	21
Lekcja 4. Deklaracje i przypisania	22
Lekcja 5. Wyprowadzanie danych na ekran	25
Lekcja 6. Operacje na zmiennych	30
Instrukcje sterujące	43
Lekcja 7. Instrukcja warunkowa if...else	43
Lekcja 8. Instrukcja switch i operator warunkowy	49
Lekcja 9. Pętle	54
Lekcja 10. Instrukcje break i continue	62
Tablice	69
Lekcja 11. Podstawowe operacje na tablicach	69
Lekcja 12. Tablice wielowymiarowe	74
Rozdział 3. Programowanie obiektowe	85
Podstawy	85
Lekcja 13. Klasy, pola i metody	86
Lekcja 14. Argumenty i przeciążanie metod	94
Lekcja 15. Konstruktory	104
Dziedziczenie	115
Lekcja 16. Klasy potomne	116
Lekcja 17. Specyfikatory dostępu i pakiety	123
Lekcja 18. Przesłanianie metod i składowe statyczne	137
Lekcja 19. Klasy i składowe finalne	149

Rozdział 4. Wyjątki	157
Lekcja 20. Blok try...catch	157
Lekcja 21. Wyjątki to obiekty	165
Lekcja 22. Własne wyjątki	173
Rozdział 5. Programowanie obiektowe II	185
Polimorfizm	185
Lekcja 23. Konwersje typów i rzutowanie obiektów	185
Lekcja 24. Późne wiązanie i wywoływanie metod klas pochodnych	194
Lekcja 25. Konstruktory oraz klasy abstrakcyjne	202
Interfejsy	211
Lekcja 26. Tworzenie interfejsów	211
Lekcja 27. Wiele interfejsów	217
Klasy wewnętrzne	225
Lekcja 28. Klasa w klasie	225
Lekcja 29. Rodzaje klas wewnętrznych i dziedziczenie	233
Lekcja 30. Klasy anonimowe i zagnieżdżone	241
Rozdział 6. System wejścia-wyjścia	249
Lekcja 31. Standardowe wejście	249
Lekcja 32. Standardowe wejście i wyjście	259
Lekcja 33. System plików	274
Lekcja 34. Operacje na plikach	285
Rozdział 7. Kontenery i typy uogólnione	301
Lekcja 35. Kontenery	301
Lekcja 36. Typy uogólnione	314
Rozdział 8. Aplikacje i aplety	327
Aplety	327
Lekcja 37. Podstawy apletów	327
Lekcja 38. Kroje pisma (fonty) i kolory	333
Lekcja 39. Grafika	342
Lekcja 40. Dźwięki i obsługa myszy	350
Aplikacje	360
Lekcja 41. Tworzenie aplikacji	360
Lekcja 42. Komponenty	376
Skorowidz	391

Rozdział 6.

System wejścia-wyjścia

Do pisania aplikacji w Javie niezbędna jest znajomość przynajmniej podstaw systemu wejścia-wyjścia. Tej tematyce jest poświęcony właśnie rozdział 6. W Javie takie operacje są wykonywane za pomocą strumieni. Strumień to abstrakcyjny ciąg danych, który działa — mówiąc w uproszczeniu — w taki sposób, że dane wprowadzone w 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 standardowo dane zostają wysłane na ekran. Strumienie wejściowe działają odwrotnie: ich początek znajduje się poza aplikacją (może być to 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 jednak zostanie omówiona jedynie komunikacja aplikacji ze światem zewnętrznym.

Lekcja 31. Standardowe wejście

Podstawowe operacje wyjściowe, czyli wyświetlanie informacji na ekranie konsoli, pojawiły się już wielokrotnie; dokładniej zostaną opisane w ramach kolejnej lekcji. Omówienie systemu wejścia-wyjścia rozpocznie się od wprowadzenia danych i standardowego strumienia wejściowego reprezentowanego przez obiekt `System.in`. Tej tematyce zostanie poświęcona cała bieżąca lekcja. Nauczysz się 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 przewidywalną 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 faktycznie odczytaną liczbę bajtów
<code>int read(byte[] b, int off, int len)</code>	Odczytuje ze strumienia liczbę bajtów nie większą niż wskazywana przez <code>len</code> i zapisuje je w tablicy <code>b</code> , poczynawszy od komórki wskazywanej przez <code>off</code> . Zwraca faktycznie 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 faktycznie 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. Spróbuj 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 również możliwe za pomocą typów `short` i `long`.

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

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

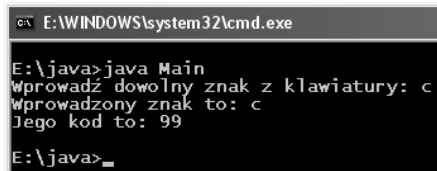
Wywołujesz zatem metodę `read` obiektu `in` zawartego w klasie `System`². Obiekt ten jest klasy `InputStream` (typem obiektu jest `InputStream`) i reprezentuje standardowy strumień wejściowy. Standardowo jest on powiązany z klawiaturą, zatem z tego strumienia będą odczytywane dane wprowadzane przez użytkownika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

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

Aby wyświetlić kod znaku, musisz ponownie dokonać konwersji. Zmienną `c` typu `char` konwertujesz zatem na typ `byte`, 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*



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 przyjmować w zależności od standardu kodowania od jednego do kilku bajtów.

Wczytywanie tekstu

Wiesz 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 by 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 finalnym i statycznym polem klasy `System`.

Zadanie to będzie wymagało skorzystania z 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 masz 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 dysponujesz. 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 fazy:

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

Znaczenie będzie takie samo, jednak w drugim przypadku zostaje utworzona zupełnie niepotrzebnie dodatkowa zmienna `isr` typu `InputStreamReader`. 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

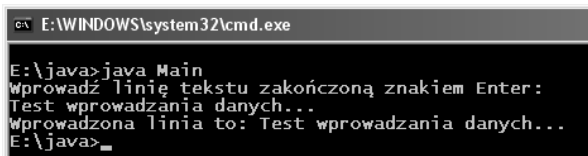
znaków) linię tekstu wprowadzoną przez użytkownika. Linia tekstu jest rozumiana jako ciąg znaków wprowadzony aż do naciśnięcia klawisza *Enter* (dokładniej: aż do osiągnięcia znaku końca wiersza w strumieniu). Wynika z tego, że instrukcja:

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

utworzy zmienną `line` typu `String` i przypisze jej obiekt 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 pokazano na rysunku 6.2.

Rysunek 6.2.

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



Skoro potrafisz już wczytać wiersz tekstu z klawiatury, spróbuj napisać program, którego zadaniem będzie wczytywanie kolejnych linii ze standardowego wejścia tak długo, aż zostanie odczytany ciąg znaków `quit`. Zadanie wydaje się banalne, wystarczy przecieć do kodu z listingu 6.2 dodać pętlę `while`, która będzie sprawdzała, czy zmienna `line` zawiera napis `quit`. W pierwszym odruchu napiszesz zapewne pętlę o następującej postaci (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 deklarujesz zmienną `line` i przypisujesz jej pusty ciąg znaków. W pętli `while` sprawdzasz 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 przypisujesz natomiast zmiennej `line` wiersz tekstu odczytany ze standardowego strumienia wejściowego oraz wyświetlasz go na ekranie.

Wszystko to wygląda bardzo poprawnie, ma tylko jedną wadę — nie zadziała zgodnie z założeniami. Spróbuj uruchomić program z listingu 6.3. Wpisz kilka linii tekstu, a następnie ciąg znaków `quit`. Teoretycznie program powinien zakończyć działanie, tymczasem działa nadal, co jest widoczne na rysunku 6.3.

Listing 6.3.

```
import java.io.*;

public class Main {
    public static void main(String args[]) {
        BufferedReader brIn = new BufferedReader(
```

Rysunek 6.3.

*Błąd w pętli while
uniemożliwia
zakończenie programu*

```

        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 zatem dzieje, gdzie jest błąd? Twoje podejrzenie powinno najpierw paść na warunek zakończenia pętli `while`. On faktycznie jest nieprawidłowy, choć wydaje się, że instrukcja ta jest poprawna. Zwróć 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 nie-nazwany obiekt klasy `String` faktycznie zawierający ciąg znaków `quit`, a w miejsce napisu `"quit"` została wstawiona referencja do tego obiektu. Jak wiesz z wcześniejszych lekcji (por. lekcja 13.), zmienna `line` jest także referencją do obiektu klasy `String`. Sama zatem instrukcja porównania `line != "quit"` porównuje ze sobą dwie REFERENCJE, które muszą być różne. Nie dochodzi tu zatem do porównania zawartości obiektów. Dlatego program nie może działać poprawnie i wpada w nieskończoną pętlę (aby go zakończyć, trzeba wcisnąć na klawiaturze kombinację klawiszy `Ctrl+C`).

Konieczne należy 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` będzie zwrócona, kiedy ciąg znaków zapisany w jednym z nich jest taki sam jak ciąg znaków zapisany w drugim. Jeśli 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`.

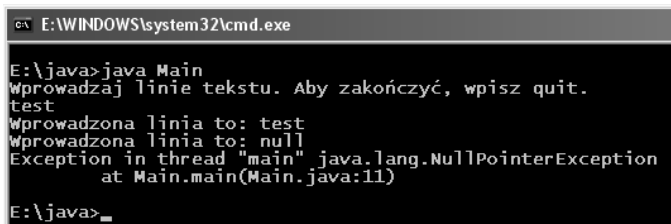
Powróćmy teraz do pętli `while`. Jako że znasz już właściwości metody `equals`, w pierwszej chwili na pewno zechcesz napisać warunek:

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

Jeśli wprowadzisz go do kodu z listingu 6.3, zacznie on poprawnie reagować na polecenie `quit` — będzie się więc wydawać, że program nareszcie działa zgodnie z założeniami. W zasadzie można by się zgodzić z tym stwierdzeniem, gdyż faktycznie działa zgodnie z założeniami, tylko że jednocześnie zawiera kolejny błąd, tym razem dużo trudniejszy do odkrycia. Ujawni się on dopiero przy próbie przerwania pracy aplikacji. Uruchom program, wpisz testową linię tekstu oraz przerwij jego działanie (w większości systemów należy wcisnąć kombinację klawiszy *Ctrl+C*). Efekt jest widoczny na rysunku 6.4. Zapewne zaskoczyło Cię powstanie wyjątku `NullPointerException`...

Rysunek 6.4.

*Ukryty błąd
w programie
spowodował
wygenerowanie
wyjątku*



```
GA E:\WINDOWS\system32\cmd.exe  
  
E:\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  
    at Main.main(Main.java:11)  
E:\java>
```

Na przyczynę powstania tego błędu naprowadza linia tekstu poprzedzająca komunikat maszyny wirtualnej, mianowicie: `Wprowadzona linia to: null`. Oznacza to bowiem (spójrz 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 masz dwie: albo dodasz w pętli warunek sprawdzający, czy `line` jest równe `null`, a jeśli tak, to przerwiesz pętlę np. instrukcją `break`; albo też zmodyfikujesz sam warunek pętli. Ta druga możliwość jest lepsza, gdyż nie powoduje wykonywania dodatkowego kodu, a jednocześnie otrzymasz ciekawą konstrukcję. Poprawiony warunek powinien bowiem wyglądać następująco:

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

Początkowo tego typu zapis budzi zdziwienie: jak można wywoływać jakąkolwiek metodę na rzecz ciągu znaków? Przypomnij sobie jednak 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, możesz wywołać metodę `equals`. Zauważ, że właśnie dzięki takiej konstrukcji unikasz wyjątku `NullPointerException`, gdyż nawet jeśli `line` będzie miało wartość `null`, to metoda `equals` po prostu zwróci wartość `false`, `null` bowiem 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.

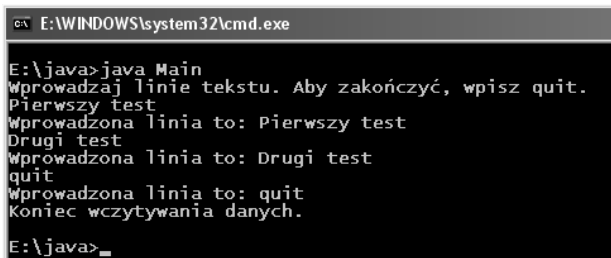
Listing 6.4.

```
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.");
        }
    }
}
```

Rysunek 6.5.

Wprowadzanie w pętli kolejnych linii tekstu



```
E:\WINDOWS\system32\cmd.exe

E:\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.
E:\java>_
```

Wprowadzanie liczb

Potrafisz już odczytywać w aplikacji wiersze tekstu wprowadzane z klawiatury, równie ważną umiejętnością jest wprowadzanie liczb. Jak to zrobić? Trzeba sobie uzmysłwić, że z klawiatury zawsze wprowadzany jest tekst. Jeśli próbujesz wprowadzić do aplika-

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

cji wartość 123, to w rzeczywistości wprowadzisz 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, możesz 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();
        }
        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);
    }
}
```

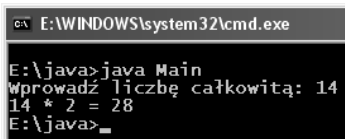
⁴ Jak widać, metody statyczne poznane w ramach lekcji 18. przydają się w praktyce.

Kod zaczynasz od utworzenia obiektu `br` w klasie `BufferedReader` powiązanego poprzez obiekt pośredniczący klasy `InputStreamReader` ze standardowym strumieniem wejściowym. Stosujesz technikę opisaną przy omawianiu przykładu z listingu 6.2. Następnie wyświetlasz prośbę o wprowadzenie liczby całkowitej oraz odczytujesz wiersz tekstu za pomocą metody `readLine` obiektu `br`. Oczywiście pamiętasz o przechwyceniu ewentualnego wyjątku `IOException`. W przypadku wystąpienia takiego wyjątku wyświetlasz stosowną informację na ekranie oraz kończysz wykonywanie programu. Zauważ, że wykorzystujesz w tym celu instrukcję `return` bez żadnych parametrów, co oznacza wyjście z funkcji `main`, a tym samym zakończenie działania aplikacji.

Jeśli odczyt wiersza tekstu się powiedzie, zostanie on zapisany w zmiennej `line`. Deklarujesz więc dalej zmienną `liczba` typu `int` oraz przypisujesz jej wynik działania metody `parseInt` z klasy `Integer`. Metodzie przekazujesz ciąg znaków zapisany w zmiennej `line`. Jeśli wprowadzony przez użytkownika ciąg znaków nie reprezentuje poprawnej wartości liczbowej, wygenerowany zostaje wyjątek `NumberFormatException`. W takim wypadku wyświetlasz komunikat o tym fakcie oraz kończysz 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`, możesz 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



```
E:\WINDOWS\system32\cmd.exe
E:\java>java Main
wprowadź liczbę całkowitą: 14
14 * 2 = 28
E:\java>
```

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

Ćwiczenia do samodzielnego wykonania

Ćwiczenie 31.1.

Napisz klasę `Main`, która będzie zawierała metodę `readLine`. Zadaniem tej metody będzie zwrócenie wprowadzonej przez użytkownika linii tekstu. Nie używaj do odczytu danych innych metod niż `System.in.read()`. Przetestuj działanie własnej metody `readLine`.

Ćwiczenie 31.2.

Zmodyfikuj kod z listingu 6.3 tak, aby warunek pętli `while` miał postać `while(!line.equals("quit"))`, ale aby nie występował błąd `NullPointerException`, kiedy osiągnięty zostanie koniec strumienia (np. po wciśnięciu kombinacji klawiszy `Ctrl+C`).

Ćwiczenie 31.3.

Napisz program (analogiczny do przedstawionego na listingu 6.5), który będzie umożliwiał wprowadzenie liczby zmiennoprzecinkowej.

Skorowidz

--, 31, 32, 33
!, 39
!=, 41
&, 38
&&, 39
., 88, 90
/* */, 19
//, 20
?:, 53
[], 70
^, 38
|, 38
||, 39
~, 38
++, 31, 32
<, 41
<<, 38
<=, 41
<applet>, 329
<object>, 329
<param>, 332
==, 41
>, 41
>=, 41
>>, 38
>>>, 38

A

abstract, 203
accept(), 280
access modifier, 123
ActionEvent, 369
ActionListener, 367, 370, 379,
380, 381, 389
actionPerformed(), 367, 368,
371, 380, 381, 389

add(), 307, 308
addActionListener(), 353, 354, 368
addDocumentListener(), 382
addItem(), 387
addKeyListener(), 376
addMouseListener(), 353, 359,
361, 376
addMouseMotionListener(),
356, 361, 376
addSeparator(), 374
addWindowListener(), 361, 362,
364
AIFF, 357
AND, 38, 39
aplety, 327
Aplet, 333
czcionki, 334
dźwięki, 350, 357
fonty, 333
getParameter(), 333
grafika, 342
init(), 331, 333
kolory, 338
konstrukcja apletu, 331
kończenie działania, 331
kroje pisma, 333, 334
obsługa myszy, 350
obsługa zdarzeń, 353
odczytywanie wartości
parametrów, 333
odświeżanie ekranu, 350
okna, 360
parametry, 332
rysowanie, 328, 329
start(), 331
stop(), 331
styl czcionki, 335

testowanie, 330
tworzenie, 328
umieszczanie na stronie, 329
uruchamianie, 331
wyświetlanie tekstu, 335
zdarzenia, 350
aplikacje, 327, 360
JFrame, 360
komponenty, 376
menu, 365
menu kontekstowe, 373
okna, 360
tworzenie, 360
append(), 270
applet, 327
Applet, 328, 333
Applet Viewer, 330
application, 327
argumenty, 94
argumenty finalne, 154
konstruktory, 105
obiekt jako argument, 96
ArithmeticException, 165,
167, 168
ArrayIndexOutOfBoundsException,
157, 161, 166, 302, 303
ArrayList, 306, 308, 310
metody, 307
ASCII, 250
AU, 357
AudioClip, 358
available(), 250

B

b-code, 14
biblioteki, 130
bity, 38

b-kod, 14
 blok try...catch, 157
 zagnieżdżanie bloków, 171
 błędy, 157
 dzielenie przez zero, 166
 boolean, 18
 break, 50, 52, 62, 255
 etykiety, 66
 BufferedInputStream, 295,
 299, 300
 BufferedOutputStream, 299, 300
 BufferedReader, 252, 260, 290,
 295, 296, 308
 buforowanie, 293
 byte, 17
 byte-code, 14

C

canExecute(), 275
 canRead(), 275
 canWrite(), 275
 case, 50, 52
 catch, 157, 162
 change code page, 28
 changedUpdate(), 382
 char, 18
 chcp, 28
 checkbox menu, 371
 CheckBoxMenu, 371
 checkError(), 270
 chmod, 11
 chronione składowe, 126
 ciągi znaków, 26
 class, 86
 ClassCastException, 315, 319
 CLASSPATH, 132
 clear(), 307
 clearError(), 270
 clearRect(), 328, 338
 close(), 250, 270, 286
 Color, 333, 338, 344
 compareTo(), 275
 Component, 376
 Console, 259, 267
 console(), 267
 contains(), 307, 376
 continue, 62, 64
 etykiety, 66, 67
 CP1250, 272
 createNewFile(), 275, 281, 282
 createTempFile(), 275
 czcionki, 28, 334
 czcionki systemowe, 334
 czcionki wbudowane, 334

czyszczenie obszaru okna
 apletu, 328

D

Dąb, 5
 default, 50
 deklaracja
 interfejsy, 211
 klasy, 86
 klasy abstrakcyjne, 203
 klasy anonimowe, 244
 klasy finalne, 149
 klasy pakietowe, 134
 klasy publiczne, 134
 klasy wewnętrzne, 225
 konstruktory, 104
 metody, 89
 metody statyczne, 144
 tablice, 70
 tablice nieregularne, 79
 typy uogólnione, 319
 wiele zmiennych, 23
 zmiennie, 22
 dekrementacja, 31, 32, 33
 Delete, 284
 delete(), 275, 283, 284
 deleteOnExit(), 275
 destruktory, 112
 Dialog, 335
 Dimension, 328
 dispose(), 363
 DivideByZeroException, 179
 do...while, 59
 DocumentListener, 382
 dołączanie pakietów, 131
 domyślne wartości dla
 poszczególnych typów
 danych, 93
 dostęp chroniony, 126
 dostęp do klasy zewnętrznej, 231
 dostęp do obiektu klasy
 wewnętrznej, 234
 dostęp prywatny, 124
 dostęp publiczny, 123
 double, 18
 Double, 258
 drawImage(), 348, 349
 drawLine(), 343
 drawOval(), 343, 344
 drawPolygon(), 343, 345
 drawPolyline(), 343
 drawRect(), 343
 drawRectangle(), 345

drawRoundRect(), 343, 345
 drawString(), 335, 336, 338, 361
 dynamic binding, 196
 dziedziczenie, 115, 116, 137
 interfejsy, 222
 klasy abstrakcyjne, 202
 klasy bazowe, 116
 klasy finalne, 149
 klasy potomne, 116
 klasy wewnętrzne, 238
 konstruktory, 119
 przesłanianie składowych, 139
 wywołanie konstruktora klasy
 bazowej, 121
 dzielenie modulo, 31
 dzielenie przez zero, 165
 dźwięki, 350, 357
 AudioClip, 358
 formaty plików
 dźwiękowych, 357
 odtwarzanie, 357, 358

E

early binding, 197
 Eclipse, 6
 edytor tekstowy, 6
 elipsy, 344
 else, 43
 empty(), 309
 enhanced for, 60
 ensureCapacity(), 307
 equals(), 254, 275
 escape sequence, 29
 etykietowany break, 66
 etykietowany continue, 66, 67
 etykiety, 66
 etykiety tekstowe, 377
 exception, 161
 Exception, 168, 173, 178
 exists(), 275, 282, 283, 289
 EXIT_ON_CLOSE, 361
 extends, 116, 222

F

false, 18, 39
 figury geometryczne, 342
 File, 266, 274, 277, 279, 297
 FileDescriptor, 297
 FileInputStream, 295
 FilenameFilter, 278, 279
 FileNotFoundException, 287,
 289, 291, 296, 297

FileOutputStream, 297
 FileReader, 295, 296
 FileWriter, 298
 fillOval(), 343, 344
 fillPolygon(), 343, 346
 fillRect(), 343
 fillRectangle(), 345
 fillRoundRect(), 343, 345
 final, 149, 150, 154
 finalize(), 112, 113
 finally, 180
 finalne argumenty, 154
 finalne klasy, 149
 finalne pola, 150
 float, 18
 Float, 258
 flush(), 268, 270
 Font, 333, 334, 336
 Font.BOLD, 335
 Font.ITALIC, 335
 Font.PLAIN, 335
 FontMetrics, 337, 338
 fonty, 333
 for, 54
 foreach, 60
 format(), 268, 270, 271
 formaty plików dźwiękowych, 357
 Frame, 360

G

garbage collector, 113
 gDC, 337
 get(), 302, 305, 307, 318
 getAbsolutePath(), 275
 getAbsolutePath(), 275
 getAllFonts(), 335, 336
 getAudioClip(), 358, 359
 getAvailableFontFamilyNames(), 335, 336
 getBackground(), 376
 getBounds(), 376
 getButton(), 352, 353
 getCanonicalFile(), 275
 getCanonicalPath(), 275
 getChannel(), 286
 getCodeBase(), 347
 getComponent(), 375
 getCursor(), 376
 getDocument(), 382
 getDocumentBase(), 347
 getFD(), 286
 getFilePointer(), 286
 getFont(), 376
 getFontMetrics(), 338, 376
 getFontName(), 336
 getForeground(), 376
 getFreeSpace(), 275
 getGraphics(), 377
 getHeight(), 337, 338, 377
 getImage(), 347
 getItemAt(), 387
 getItemCount(), 387
 getLocalGraphicsEnvironment(), 336
 getMessage(), 167
 getModifiers(), 356
 getName(), 275, 377
 getParameter(), 333, 341
 getParent(), 275, 377
 getParentFile(), 275
 getPath(), 275, 284
 getSelectedIndex(), 387, 389
 getSelectedItem(), 387, 389
 getSize(), 328, 377
 getSource(), 369, 373
 getText(), 377, 382
 getTotalSpace(), 276
 getUsableSpace(), 276
 getWidth(), 377
 getX(), 352, 356, 377
 getY(), 352, 356, 377
 GIF, 347
 grafika, 328, 342
 elipsy, 344
 figury geometryczne, 342
 Graphics, 342
 koła, 344
 linie, 342
 obrazy, 347
 odświeżanie ekranu, 350
 prostokąt o zaokrąglonych rogach, 345
 punkty, 342
 rysowanie, 342
 wczytywanie grafiki, 347
 wielokąty, 345
 Graphics, 328, 338, 342
 metody, 343
 GraphicsEnvironment, 335, 336
 GridLayout, 386

H

hashCode(), 276
 hasNext(), 266, 311, 313
 hasNextByte(), 266

hasNextDouble(), 266
 hasNextInt(), 266, 267
 hasNextLine(), 266
 heap, 88
 height, 328
 hierarchia klas, 213
 hierarchia wyjątków, 168
 historia Javy, 5
 HTML, 329, 330

I

if...else, 43
 if...else if, 46
 iloczyn logiczny, 39
 ImageObserver, 342, 348, 349, 350
 ImageObserver.ALLBITS, 350
 imageUpdate(), 349
 implementacja interfejsu, 212
 implementacja wielu interfejsów, 217
 implements, 212, 215, 217
 import, 131, 268
 indeksowanie tablicy, 157
 indexOf(), 307
 inicjalizacja, 22
 obiekty, 104
 pola finalne, 152
 tablice, 72
 init(), 331, 333
 initialization, 22
 initiation, 22
 inkrementacja, 31, 32
 InputEvent, 356
 InputReader, 266
 InputStream, 249, 250, 251, 252, 267
 InputStreamReader, 252
 insertItemAt(), 387
 insertUpdate(), 382
 instalacja JDK, 9
 Linux, 10
 Windows, 10
 instrukcje, 21
 instrukcje sterujące, 43
 int, 17, 35
 Integer, 257
 interface, 211
 interfejsy, 211
 ActionListener, 367, 381
 AudioClip, 358
 deklaracja, 211
 DocumentListener, 382

- interfejsy
 - dziedziczenie, 222
 - FilenameFilter, 279
 - hierarchia klas, 213
 - ImageObserver, 349
 - implementacja, 212
 - ItemListener, 371
 - Iterable, 311
 - konflikty nazw, 219, 223
 - MouseListener, 351, 353
 - MouseMotionListener, 355
 - pola, 216
 - Readable, 266
 - ReadableByteChannel, 266
 - Runnable, 361
 - WindowListener, 361
 - invokeLater(), 361
 - IOException, 251, 258, 261, 282
 - isAbsolute(), 276
 - isDirectory(), 276
 - isEditable(), 387
 - isEmpty(), 307
 - isFile(), 276
 - isHidden(), 276
 - isPopupTrigger(), 374
 - ItemListener, 371, 373
 - itemStateChanged(), 371, 373
 - Iterable, 311, 312
 - metody, 311
 - iteracja po kolekcji, 60
 - iterator(), 311, 312
 - iteratory, 311
- J**
- JApplet, 328
 - java, 14
 - Java, 5
 - Java 2, 7
 - Java 2 Platform, 7
 - Java 6, 7
 - Java 7, 7
 - java compiler, 13
 - Java Development Kit, 6, 9
 - Java Enterprise Edition, 7
 - Java Mobile Edition, 7
 - Java Runtime Environment, 6, 10
 - Java SE 6, 7
 - Java Standard Edition, 7
 - Java Virtual Machine, 14
 - java.awt, 328, 334, 360
 - java.awt.event, 351
 - java.io, 251
 - java.util.regex, 280
 - javac, 13
 - javax.swing, 328, 360
 - javax.swing.event, 364
 - jawnna konwersja, 186
 - JButton, 379
 - JCheckBox, 385
 - konstruktory, 385
 - JCheckBoxMenuItem, 371, 373
 - JComboBox, 387
 - konstruktory, 387
 - metody, 387
 - JComponent, 376, 380
 - JDK, 6
 - instalacja, 9
 - przygotowanie do pracy, 11
 - JEditorPane, 380
 - jednostka kompilacji, 134
 - jednowierszowe pole tekstowe, 380
 - język Java, 5
 - JFormattedTextField, 380
 - JFrame, 360
 - JLabel, 377
 - JMenu, 365, 369
 - JMenuBar, 365, 366
 - JMenuItem, 365, 366, 367, 371, 374
 - JPasswordField, 380
 - JPG, 347
 - JPopupMenu, 373, 374
 - JRE, 6, 10
 - JTextArea, 380, 383
 - konstruktory, 383
 - JTextComponent, 380
 - JTextField, 380, 382
 - konstruktory, 381
 - JTextPane, 380
- K**
- kaskadowe menu, 369
 - katalogi, 277
 - tworzenie, 281, 282
 - usuwanie, 283
 - klasy, 15, 16, 86
 - ActionEvent, 369
 - Applet, 328, 333
 - ArrayList, 306
 - BufferedInputStream, 299
 - BufferedOutputStream, 299
 - BufferedReader, 252, 260, 290
 - CheckBoxMenu, 371
 - Color, 338
 - Component, 376
 - Console, 259, 267
 - definicja, 86
 - dziedziczenie, 115, 116
 - Exception, 168, 173
 - File, 274
 - FileInputStream, 295
 - FileReader, 295
 - Font, 333, 334
 - FontMetrics, 337
 - Graphics, 328, 342
 - GraphicsEnvironment, 335, 336
 - GridLayout, 386
 - hierarchia klas, 213
 - ImageObserver, 350
 - implementacja interfejsu, 212
 - implementacja wielu interfejsów, 217
 - inicjalizacja pól finalnych, 152
 - InputEvent, 356
 - InputStream, 250, 251
 - InputStreamReader, 252
 - Integer, 257
 - JApplet, 328
 - JButton, 379
 - JCheckBox, 385
 - JCheckBoxMenuItem, 371
 - JComboBox, 387
 - JComponent, 376
 - JEditorPane, 380
 - JFormattedTextField, 380
 - JFrame, 360
 - JLabel, 377
 - JMenu, 365
 - JMenuBar, 365, 366
 - JMenuItem, 365, 366
 - JPasswordField, 380
 - JPopupMenu, 373
 - JTextArea, 380, 383
 - JTextField, 380
 - JTextPane, 380
 - klasy bazowe, 116
 - klasy finalne, 149
 - klasy kontenerowe, 301, 305
 - klasy nadrzędne, 116
 - klasy pakietowe, 134, 236
 - klasy podrzędne, 116
 - klasy potomne, 116
 - klasy publiczne, 134
 - klasy statyczne, 246
 - klasy zagnieżdżone, 241, 246
 - klasy zewnętrzne, 231
 - konstruktory, 104
 - metody, 89
 - metody finalne, 154

- metody prywatne, 199, 200
 - metody statyczne, 144
 - MouseAdapter, 374
 - MouseEvent, 351, 356
 - MouseInputAdapter, 364, 365
 - nazwy, 87
 - Object, 191
 - odwołanie do składowych, 88
 - OutputStreamWriter, 272, 297
 - pakiety, 130
 - Pattern, 280
 - pola, 86, 87
 - pola statyczne, 146
 - Polygon, 346
 - PrintStream, 270
 - PrintWriter, 272, 273
 - RandomAccessFile, 285
 - RuntimeException, 168
 - Scanner, 259, 266
 - składowe, 86
 - składowe statyczne, 144
 - specyfikatory dostępu, 123
 - Stack, 308
 - StreamTokenizer, 259
 - String, 254
 - SwingUtilities, 361
 - System, 249
 - Throwable, 168, 173
 - tworzenie obiektu, 87
 - URL, 347, 357
 - wartości domyślne pól, 92
 - WindowAdapter, 363
 - wyjątki, 168
 - klasy abstrakcyjne, 202
 - definicja, 203
 - klasy anonimowe, 241, 243
 - obsługa zdarzeń, 363
 - tworzenie, 243
 - klasy wewnętrzne, 225
 - deklaracja, 225
 - dostęp do klasy
 - zewnętrznej, 231
 - dziedziczenie z klasy
 - zewnętrznej, 238
 - kilka klas wewnętrznych, 227
 - klasy chronione, 237
 - klasy pakietowe, 237
 - klasy prywatne, 237
 - klasy publiczne, 237
 - obiekty, 233
 - rodzaje klas wewnętrznych, 236
 - składowe, 228
 - tworzenie, 225
 - umieszczenie klasy
 - wewnętrznej, 230
 - kod ASCII, 250
 - kod HTML, 330
 - kod pośredni, 14
 - kodowanie znaków, 27, 271
 - kolekcje, 60
 - kolory, 338, 344
 - RGB, 340
 - koła, 344
 - komentarze, 18
 - komentarz blokowy, 19
 - komentarz liniowy (wierszowy), 20
 - kompilacja, 13
 - kompilator, 13
 - komponenty, 376
 - Component, 376
 - etykiety, 377
 - JButton, 379
 - JCheckBox, 385
 - JComboBox, 387
 - JComponent, 376
 - JEditorPane, 380
 - JFormattedTextField, 380
 - JLabel, 377
 - JPasswordField, 380
 - JTextArea, 380, 383
 - JTextField, 380
 - JTextPane, 380
 - listy rozwijane, 387
 - menedżery układu, 386
 - pola tekstowe, 380
 - pola wyboru, 385
 - przyciski, 379
 - rozkład elementów w oknie, 379
 - Swing, 328
 - komunikat o wyjątku, 167
 - konflikty nazw, 219, 223
 - konsola, 12, 267
 - konstruktory, 104, 202
 - argumenty, 105
 - deklaracja, 104
 - dziedziczenie, 119
 - klasy bazowe, 119
 - klasy potomne, 119
 - konstruktor domyślny, 207
 - odwołanie do obiektu
 - bieżącego, 108
 - przeciążanie, 106
 - this, 108
 - wywołanie, 205
 - wywołanie innego konstruktora, 110
 - wywołanie konstruktora klasy bazowej, 121
 - wywołanie metod, 109, 209
 - kontenery, 301
 - ArrayList, 306
 - iteratory, 311
 - klasy kontenerowe, 305
 - listy, 306
 - przeglądanie kontenerów, 310
 - Stack, 308
 - stos, 308
 - stosowanie, 317
 - kontrola typów, 314
 - konwersja typów, 35, 185
 - konwersja typów prostych, 186
 - konwersje automatyczne, 35
 - kopiowanie plików, 291, 299
 - kroje pisma, 333, 334
- ## L
- labeled break, 66
 - labeled continue, 66
 - lastIndexOf(), 307
 - lastModified(), 276
 - late binding, 196
 - length, 72
 - tablice wielowymiarowe, 77
 - length(), 276, 286
 - liczby całkowite, 17, 35
 - liczby zmiennoznakowe, 18
 - linie, 342
 - Linux, 10
 - list(), 276, 277, 280
 - listFiles(), 276
 - listRoots(), 276
 - listy, 306
 - listy rozwijane, 387
 - logiczna negacja, 40
 - logiczna suma, 39
 - logiczny iloczyn, 39
 - long, 17
 - loop(), 358
- ## M
- main(), 99, 101, 145
 - Main.class, 14
 - Main.java, 13
 - mark(), 250
 - markSupported(), 250
 - matcher(), 280
 - matches(), 281
 - menu, 365

menu kaskadowe, 369
 menu kontekstowe, 373
 metody, 89
 addActionListener(), 353, 354
 argumenty, 94
 argumenty finalne, 154
 deklaracja, 89
 dispose(), 363
 drawString(), 335, 336
 equals(), 254
 finalize(), 112, 113
 get(), 302, 318
 getParameter(), 333
 hasNext(), 313
 init(), 331, 333
 iterator(), 311
 konstruktor, 104
 main(), 99, 101, 145
 metody abstrakcyjne, 204
 metody finalne, 154
 metody prywatne, 199
 next(), 313
 obiekt jako argument, 96
 paint(), 328, 333, 336
 przeciążanie, 102
 przesłanianie, 137, 139
 repaint(), 350, 354
 return, 91
 set(), 302, 304, 318
 setFont(), 335
 specyfikatory dostępu, 124
 start(), 331
 stop(), 331
 super(), 121
 toString(), 192
 typy uogólnione, 322
 wartość zwracana, 89
 wyjątki, 173
 wywołanie, 90
 zwracanie wartości, 91
 zwracanie wielu wartości, 323
 metody statyczne, 144
 deklaracja, 144
 wywołanie, 145
 MIDI, 357
 mkdir(), 276, 282
 mkdirs(), 276, 282
 modyfikatory dostępu, 123
 MouseAdapter, 374
 mouseClicked(), 351, 353
 mouseDragged(), 355, 356
 mouseEntered(), 351, 353
 MouseEvent, 351, 356, 375

MouseEvent.ALT_DOWN_MASK, 356
 MouseEvent.BUTTON1, 352
 MouseEvent.BUTTON2, 352
 MouseEvent.BUTTON3, 352
 MouseEvent.CTRL_DOWN_MASK, 356
 MouseEvent.NOBUTTON, 352
 MouseEvent.SHIFT_DOWN, 356
 MouseEvent.SHIFT_DOWN_MASK, 356
 mouseExited(), 351, 353
 MouseInputAdapter, 364, 365
 MouseListener, 351, 353, 359, 364, 374
 metody, 351
 MouseMotionListener, 355
 mouseMoved(), 355, 356
 mousePressed(), 351, 353, 374
 mouseReleased(), 351, 353, 374
 mysz, 350, 364

N

narzędzia, 6
 nazwy
 klasy, 15, 87
 pakiety, 131
 pliki źródłowe, 15
 zmienne, 22, 24
 negacja logiczna, 40
 NetBeans, 6
 new, 70, 87, 118
 next(), 311, 313
 nextByte(), 266
 nextDouble(), 266
 nextInt(), 266
 nextLine(), 266
 nextToken(), 263
 niszczenie obiektów, 113
 NOT, 38, 40
 null, 25, 87
 NullPointerException, 171, 255
 NumberFormatException, 257, 341

O

Oak, 5
 obiektowość, 15
 obiekty, 16, 104
 destruktor, 112
 inicjalizacja, 104

inicjalizacja pól finalnych, 152
 klasy wewnętrzne, 233
 konstruktor, 104
 metody, 89
 metody finalne, 154
 porównywanie obiektów, 254
 tworzenie, 87
 typy uogólnione, 321
 wyjątki, 165
 Object, 168, 191, 323
 object types, 24
 obrazy graficzne, 347
 obsługa konsoli, 267
 obsługa myszy, 350
 MouseEvent, 351, 356
 MouseInputAdapter, 364
 MouseListener, 351, 353
 MouseMotionListener, 355
 parametry zdarzenia, 356
 współrzędne położenia kursora myszy, 355
 zdarzenia, 351
 obsługa wyjątków, 166
 obsługa zdarzeń, 353, 361
 klasy anonimowe, 363
 odczyt bajtów ze strumienia, 250
 odczyt danych z plików, 288, 295
 odświeczacz, 113
 odświeżanie ekranu, 350
 odtwarzanie klipu dźwiękowego, 357
 odwołanie do obiektu bieżącego, 108
 odwołanie do przesłoniętego pola, 143
 odwołanie do składowych klasy, 88
 odwrócona nazwa domeny, 131
 okna, 360
 działanie wykonywane podczas zamykania, 361
 JFrame, 360
 menedżery układu, 386
 rozkład elementów, 379
 rozmiar, 360
 WindowListener, 362
 wyświetlanie, 361
 zdarzenia, 361
 OpenJDK, 6
 operacje graficzne, 328
 operacje na bitach, 38
 operacje na plikach, 285, 287
 operacje na tablicach, 69
 operacje na zmiennych, 30

operacje wyjściowe, 249
operator, 30
 new, 70
 operator warunkowy, 53
 operatory arytmetyczne, 31
 operatory bitowe, 38
 operatory logiczne, 39
 operatory porównywania, 41
 operatory relacyjne, 41
 priorytety operatorów, 42
 przypisanie, 40
OR, 38, 39
OutputStreamWriter, 272, 273,
 297
overloading, 102
overriding, 139

P

package, 130
paint(), 328, 333, 336
pakiet JDK, 6
pakiety, 130
 nazwy, 131
 odwrócona nazwa domeny, 131
 stosowanie, 132
 tworzenie, 130
parametry, 94, 96
parametry apletu, 332
parametry wiersza poleceń, 100
parametry wywołania
 programu, 100
parametry zdarzenia, 356
parseDouble(), 258
parseFloat(), 258
parseInt(), 257, 258
pasek menu, 365
PATH, 12
Pattern, 280
peek(), 309
pętla, 54
 break, 62
 continue, 62, 64
 do...while, 59
 for, 54
 foreach, 60
 przejdź do kolejnej iteracji, 64
 przerwanie wykonania, 62
 while, 57
play(), 357
pliki, 274, 281
 kopiowanie, 291, 299
 odczyt danych, 288, 295
 operacje, 285

operacje strumieniowe, 294
pliki o dostępie swobodnym,
 285
 tryb otwarcia, 287
 tworzenie, 281
 usuwanie, 283
 zapis danych, 289, 296
PNG, 347
pobieranie zawartości katalogu,
 277
pola, 86, 87
 interfejsy, 216
 pola statyczne, 146
 przesłanianie, 139, 142
 wartości domyślne, 92
pola finalne, 150
 inicjalizacja, 152
 pola typów prostych, 150
 pola typów referencyjnych, 151
pola tekstowe, 380
pola wyboru, 385
polimorficzne wywoływanie
 metod, 210
polimorfizm, 185, 196
polskie znaki, 27, 271
Polygon, 346
ponowne zgłoszenie
 przechwyconego wyjątku, 175
pop(), 308, 309
porównania, 41
 porównanie obiektów, 254
późne wiązanie, 194, 196
print(), 271
printf(), 268, 269, 271
 znaczniki formatujące, 269
println(), 251, 271
PrintStream, 270, 272
PrintWriter, 272, 273
priorytety operatorów, 42
private, 124, 127
program, 13, 360
 parametry wywołania, 100
 punkt startowy, 99
 struktura, 13
 uruchamianie, 14
programowanie, 9, 13
programowanie obiektowe, 16,
 85, 185
prostokąt o zaokrąglonych
 rogach, 345
protected, 126, 127
prywatne składowe, 124
przechowywanie wielu
 danych, 301

przechwytywanie wyjątków, 157
 przechwytywanie wielu
 wyjątków, 169
przeciążanie
 konstruktory, 106
 metody, 94, 102, 137
przeglądanie kontenerów, 310
przekroczenie zakresu
 wartości, 37
przesłanianie
 metody, 137, 139
 pola, 139, 142
przesunięcie bitowe, 38
przetwarzanie kolekcji, 60
przyciski, 379
przypisanie, 22, 40
public, 100, 123
publiczne składowe, 123
punkt startowy programu, 99
punkty, 342
push(), 308, 309, 310

R

RandomAccessFile, 285, 287,
 289
 kopiowanie plików, 291
 odczyt danych z pliku, 288
 operacje na plikach, 287
 zapis danych do pliku, 289
read(), 250, 251, 286, 294, 295
Readable, 266
ReadableByteChannel, 266
readBoolean(), 286
readByte(), 286
readChar(), 286
readDouble(), 286
reader(), 268
readFloat(), 286
readFully(), 286
readInt(), 286
readLine(), 251, 252, 253, 259,
 268, 286, 289
readLong(), 286
readPassword(), 268
readShort(), 286
readUnsignedByte(), 286
readUnsignedShort(), 286
readUTF(), 286
reference types, 24
referencje, 25, 152
remove(), 307, 311
removeAllItems(), 387
removeItem(), 387

removeItemAt(), 387
 removeRange(), 307
 removeUpdate(), 382
 renameTo(), 276
 repaint(), 350, 354, 377
 reset(), 250
 resize(), 303, 305
 return, 91, 160, 282
 RGB, 340
 rodzaje klas wewnętrznych, 236
 rozkład elementów w oknie, 379
 rozmiar tablicy, 72
 run(), 361
 Runnable, 361
 runtime binding, 196
 RuntimeException, 168, 169
 rysowanie, 328, 329

- figury geometryczne, 342
- kolory, 344

 rzeczywisty typ obiektu, 194
 rzutowanie na interfejs, 243
 rzutowanie na typ Object, 191
 rzutowanie typów obiektowych, 185, 187, 190, 241
 rzutowanie w górę, 196, 319

S

Scanner, 259, 266, 267
 search(), 309
 seek(), 286
 sekwencje ucieczki, 29
 sekwencje znaków specjalnych, 29
 set(), 304, 305, 307, 318
 setBackground(), 377
 setBounds(), 377, 378
 setColor(), 328, 338, 344
 setCursor(), 377
 setDefaultCloseOperation(), 361
 setEditable(), 387
 setError(), 271
 setExecutable(), 276
 setFont(), 335, 337, 377
 setMenuBar(), 365
 setLastModified(), 276
 setLayout(), 379, 386
 setLength(), 286
 setLocation(), 377
 setName(), 377
 setReadable(), 276, 277
 setReadOnly(), 277
 setSelectedIndex(), 387
 setSelectedItem(), 387

setSize(), 360, 377
 setText(), 377, 378, 379, 380
 setTitle(), 382
 setVisible(), 361, 377
 setWritable(), 277
 short, 17
 size(), 303, 305, 307
 skip(), 250
 składowe, 86

- składowe klas wewnętrznych, 228
- składowe statyczne, 144
- specyfikatory dostępu, 123

 specyfikatory dostępu, 123

- private, 124
- protected, 126
- public, 123

 sprawdzanie poprawności danych, 157
 stack, 88
 Stack, 308, 310

- metody, 309

 stałe kolorów, 339
 standardowe wejście, 249, 259
 standardowe wyjście, 259
 standardowy strumień

- wejściowy, 249

 start(), 331, 358
 static, 100, 144, 146
 statyczne pola, 146
 sterta, 88
 stop(), 331, 358
 stos, 88, 308
 StreamTokenizer, 259, 261, 263

- ttype, 260

 StreamTokenizer.TT_EOF, 260
 StreamTokenizer.TT_EOL, 260
 StreamTokenizer.TT_NUMBER, 260
 StreamTokenizer.TT_WORD, 260
 String, 100, 254, 266
 string(), 338
 stringWidth(), 337
 strona HTML, 329
 struktura programu, 13
 struktury danych, 69, 301
 strumienie, 249

- BufferedInputStream, 299
- BufferedOutputStream, 299
- BufferedReader, 252
- buforowanie, 299
- InputStream, 250, 251
- InputStreamReader, 252

odczyt bajtów, 250
 OutputStreamWriter, 272
 PrintStream, 270
 PrintWriter, 272, 273
 standardowe wejście, 259
 standardowe wyjście, 259
 standardowy strumień

- wejściowy, 249

 System.in, 249
 System.out, 249
 wczytywanie tekstu, 251
 wprowadzanie liczb, 256
 wyjście, 249
 strumieniowe operacje na plikach, 294
 FileInputStream, 295
 kopiowanie plików, 299
 odczyt danych, 295
 zapis do pliku, 296
 styl czcionki, 335
 suma logiczna, 39
 Sun Microsystems, 5
 super(), 121, 140
 Swing, 328

- obsługa zdarzeń, 361

 SwingUtilities, 361
 switch, 49, 50, 52

- break, 50, 52
- case, 50, 52
- default, 50

 System, 249
 system dwójkowy, 38
 system dziesiętny, 38
 system plików, 274

- File, 274

 katalogi, 277, 281
 pliki, 281
 pobieranie zawartości katalogu, 277
 usuwanie katalogów, 283
 usuwanie plików, 283
 system wejścia-wyjścia, 249
 System.console(), 267
 System.gc(), 114
 System.in, 249
 System.out, 249, 273
 System.out.print(), 30
 System.out.println(), 25, 26, 193, 251, 296

Ś

środowisko programistyczne, 6

T

tablice, 69
 deklaracja, 70
 elementy, 71
 indeksy, 71, 157
 inicjalizacja, 72
 length, 72
 odwołanie do nieistniejącego elementu, 158
 operacje, 69
 rozmiar tablicy, 72
 tworzenie, 70
 tablice nieregularne, 79
 deklaracja, 79
 wyświetlanie danych, 81
 tablice wielowymiarowe, 74
 length, 77
 tablice dwuwymiarowe, 74
 tworzenie, 75
 wyświetlanie zawartości, 76
 testowanie apletów, 330
 this, 108, 110, 111
 throw, 173, 174, 175, 302
 Throwable, 168, 173
 throws, 173
 toArray(), 307
 tokeny, 259
 toString(), 192, 277
 toURI(), 277
 toURL(), 277
 true, 18, 39
 True Type, 28
 try...catch, 157, 162
 tworzenie
 aplety, 328
 aplikacje, 360
 interfejsy, 211
 katalogi, 281, 282
 klasy anonimowe, 243
 klasy wewnętrzne, 225
 listy, 306
 menu, 365
 menu kontekstowe, 374
 obiekty, 87
 obiekty klas wewnętrznych, 235
 pakiety, 130
 pliki, 281
 tablice, 70
 tablice nieregularne, 80
 tablice wielowymiarowe, 75
 wyjątki, 178
 typ obiektu, 16

type(), 321, 324
 typy danych, 15, 17
 arytmetyczne
 całkowitoliczbowe, 17
 arytmetyczne
 zmiennopozycyjne, 18
 boolean, 18
 byte, 17
 char, 18
 domyślne wartości, 93
 double, 18
 float, 18
 int, 17
 konwersja, 35, 185
 long, 17
 short, 17
 znaki, 18
 typy referencyjne, 24
 typy uogólnione, 301, 314
 definicja, 319
 metody, 322
 stosowanie, 318
 tworzenie obiektu, 321

U

ukrywanie wnętrza klasy, 127
 umiejscowienie klasy
 wewnętrznej, 230
 umieszczanie apletów na stronie, 329
 Unicode, 28
 UnsupportedOperationException, 272
 UnsupportedOperationException, 313
 uogólnianie metod, 322
 URL, 347, 357
 uruchamianie apletu, 331
 uruchamianie programu, 14
 usuwanie
 katalogi, 283
 obiekty, 113
 pliki, 283

V

void, 89, 90, 94

W

wartości domyślne pól, 92
 wartości null, 87
 wartość pusta, 25

WAVE, 357
 wbudowane typy danych, 17
 wczesne wiązanie, 197
 wczytywanie
 grafika, 347
 tekst, 251
 wejście, 249
 wersje Javy, 7
 wersje JDK, 7
 wewnętrzne klasy statyczne, 246
 while, 57, 253, 254
 wiązanie czasu wykonania, 196
 wiązanie dynamiczne, 196
 width, 328
 wielokąty, 345
 windowActivated(), 362
 WindowAdapter, 363
 windowClosed(), 362
 windowClosing(), 362, 363
 windowDeactivated(), 362
 windowDeiconified(), 362
 windowIconified(), 362
 WindowListener, 361, 362, 363
 metody, 362
 windowOpened(), 362
 Windows, 10
 wirtualna maszyna Javy, 14
 własne wyjątki, 173
 wprowadzanie danych w
 rzeczywistych aplikacjach, 262
 wprowadzanie liczb, 256
 write(), 271, 286, 297
 writeBoolean(), 286
 writeByte(), 287
 writeBytes(), 287, 291
 writeChar(), 287
 writeChars(), 287
 writeDouble(), 287
 writeFloat(), 287
 writeInt(), 287, 289
 writeLong(), 287
 writer(), 268
 writeShort(), 287
 writeUTF(), 287
 współrzędne położenia kursora
 myszy, 355
 wyjątki, 157, 161
 ArithmeticException, 165
 ArrayIndexOutOfBoundsException, 161, 302
 ClassCastException, 315
 Exception, 168, 173, 178
 FileNotFoundException, 287

- hierarchia wyjątków, 168
- IOException, 251, 258, 282
- klasy, 168
- komunikat o wyjątku, 167
- NullPointerException, 171, 255
- NumberFormatException, 257, 341
- obiekty, 165, 166
- ponowne zgłoszenie
 - przechwyconego wyjątku, 175
- przechwytywanie, 157
- przechwytywanie wielu wyjątków, 169
- RuntimeException, 168
- sekcja finally, 180
- throw, 173, 175
- Throwable, 168, 173
- try...catch, 157, 162
- tworzenie wyjątków, 178
- UnsupportedEncoding
 - ↳Exception, 272
- UnsupportedOperation
 - ↳Exception, 313

- wiele bloków catch, 169
- własne wyjątki, 173
- zagnieżdżanie bloków
 - try...catch, 171
- zgłaszanie, 173
- wyjście, 249, 270
- wykonanie programu, 13
- wyprowadzanie danych na ekran, 25
- wrażenia regularne, 280
- wyświetlanie informacji na ekranie, 249
- wyświetlanie tekstu, 335
- wyświetlanie wartości zmiennych, 25
- wywołanie
 - konstruktor klasy bazowej, 121
 - konstruktory, 205
 - metody, 90, 197
 - metody klas pochodnych, 199
 - metody przesłonięte, 140
 - metody przez konstruktory, 109, 209
 - metody statyczne, 145

Z

- zagnieżdżanie bloków
 - try...catch, 171
- zapis danych do pliku, 289, 296
- zarządzanie pamięcią, 113
- zdarzenia, 350
 - obsługa, 353, 363
 - okna, 361
 - parametry, 356
- zgłaszanie wyjątków, 173
- zmiana koloru, 338
- zmienne, 21
 - deklaracja, 22
 - deklaracja wielu zmiennych, 23
 - inicjalizacja, 22
 - nazwy, 22, 24
 - operacje, 30
 - przypisanie wartości, 22
 - typy odnośnikowe, 24
 - zmienne odnośnikowe, 87
 - zmienne referencyjne, 24
- znaki, 18
- znaki specjalne, 28

X

- XOR, 38

Język Java nieprzerwanie święci triumfy na salonach profesjonalnych firm zajmujących się programowaniem. Jest wykorzystywany zarówno w prostych programach dla telefonów komórkowych, jak i w skomplikowanych aplikacjach sieciowych. Jego główne zalety to duża przenośność i świetna, przemyślana konstrukcja, która pozwala łatwo opanować zasady programowania i szybko zacząć tworzyć własne, dobrze działające programy. Java ma jeszcze jedną cechę, istotną z punktu widzenia każdej osoby zajmującej się informatyką — po prostu nie wypada jej nie znać!

Książka „Praktyczny kurs Java, Wydanie III” oferuje swoim czytelnikom możliwość łatwego i szybkiego zapoznania się z podstawami programowania w tym języku. Z jej pomocą w mig zainstalujesz odpowiednie środowisko programistyczne i poznasz reguły budowania aplikacji w Javie, typy danych oraz rodzaje zmiennych. Nauczysz się kontrolować przebieg wykonywania programu oraz wykorzystywać tablice. Zrozumiesz, na czym polega programowanie obiektowe i związane z nim podstawowe pojęcia, takie jak dziedziczenie i polimorfizm. Dowiesz się, jak obsługiwać i tworzyć wyjątki, jak działa system wejścia-wyjścia, co to są kontenery i typy uogólnione oraz czym różnią się aplikacje od apletów. A wszystko to w serii znakomitych, praktycznych ćwiczeń!

- **Krótką historią Javy, jej narzędzia i wersje**
- **Instalacja JDK i podstawy programowania**
- **Zmienne, instrukcje sterujące i tablice**
- **Dziedziczenie, polimorfizm, interfejsy i klasy wewnętrzne**
- **Wyjątki**
- **System wejścia-wyjścia**
- **Kontenery i typy uogólnione**
- **Aplikacje i aplety**

Zanurz się w świecie Javy!

helion.pl
księgarnia
internetowa



Helion

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

📖 Książki najchętniej czytane:

🔗 <http://helion.pl/bestsellery>

📖 Zamów informacje o nowościach:

🔗 <http://helion.pl/novosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

Nr katalogowy: **6124**



Księgarnia internetowa:

<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900

Cena 59,00 zł

ISBN 978-83-246-3228-2



Informatyka w najlepszym wydaniu