# 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 aplety

Marcin Lis

Helion

Wydanie III

# 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 aplety

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!

- 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 aplety

**Zanurz się w świecie Javy!**



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

# Spis treści

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 |
|---|---|
| `int available()` | Zwraca przewidywalną liczbę bajtów, które mogą być pobrane ze strumienia przy najbliższym odczycie |
| `void close()` | Zamyka strumień i zwalnia związane z nim zasoby |
| `void mark(int readlimit)` | Zaznacza bieżącą pozycję w strumieniu |
| `boolean markSupported()` | Sprawdza, czy strumień może obsługiwać metody `mark` i `reset` |
| `abstract int read()` | Odczytuje kolejny bajt ze strumienia |
| `int read(byte[] b)` | Odczytuje ze strumienia liczbę bajtów nie większą niż rozmiar tablicy `b`. Zwraca faktycznie odczytaną liczbę bajtów |
| `int read(byte[] b, int off, int len)` | Odczytuje ze strumienia liczbę bajtów nie większą niż wskazywana przez `len` i zapisuje je w tablicy `b`, począwszy od komórki wskazywanej przez `off`. Zwraca faktycznie przeczytaną liczbę bajtów |
| `void reset()` | Wraca do pozycji strumienia wskazywanej przez wywołanie metody `mark` |
| `long skip(long n)` | Pomija w strumieniu liczbę bajtów wskazywanych przez `n`. 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 +127[1]. 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:

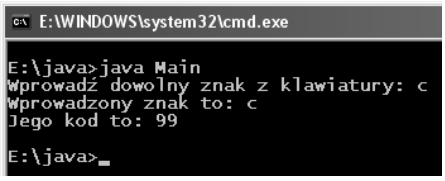
```
char c = (char) System.in.read();
```

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 `Input ↪Stream`), 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);
```

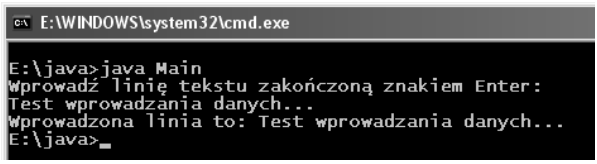
Znaczenie będzie takie samo, jednak w drugim przypadku zostaje utworzona zupełnie niepotrzebnie dodatkowa zmienna `isr` typu `InputStreamReader`. Który ze sposobów jest czytelniejszy i zostanie zastosowany, zależy jednak od indywidualnych preferencji programisty. Z reguły stosuje się sposób pierwszy.

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 przecież 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.");
```

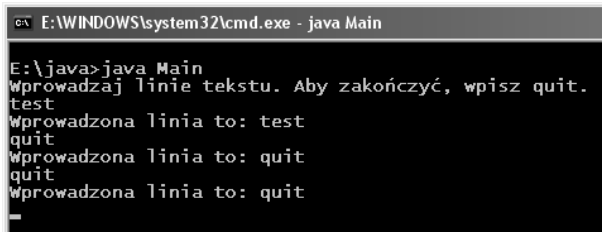
Warto przeanalizować ten przykład. Na początku deklarujesz zmienną `line` i przypisujesz jej pusty ciąg znaków. W pętli `while` sprawdzasz warunek, czy zmienna `line` jest różna od ciągu znaków `quit` — jeśli tak, to pętla jest kontynuowana, a jeśli nie, to kończy działanie. Zatem znaczenie jest następujące: 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 nienazwany 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*).

Koniecznie 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:

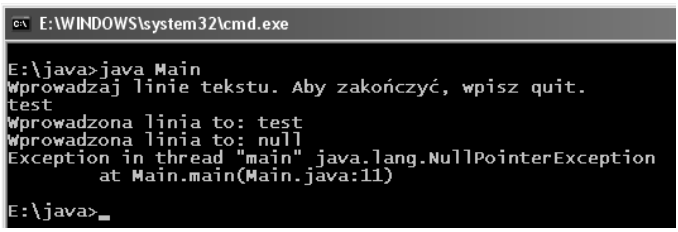
```
napis1.equals("java");
```

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*

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:

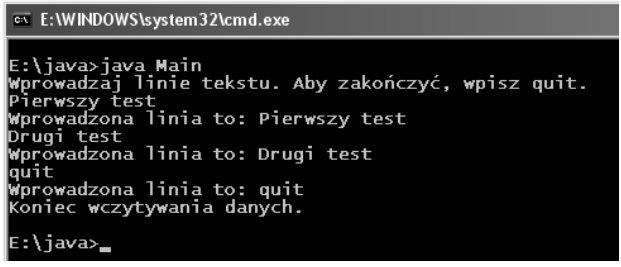
```
while(!"quit".equals(line)){
  /*instrukcje pętli while*/
}
```

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 literał[3] `"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*




## Wprowadzanie liczb

Potrafisz 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 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.

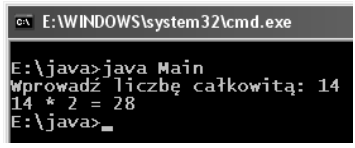
**Listing 6.5.**

```
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 `brIn` klasy `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 `brIn`. 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*

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

## C



## D



## E



## F

## G



## H



## I

## J



## K

## L



## M

## N



## O

## P



## R

## S



## Ś

## T



## U



## V



## W

## X



## Z

# Praktyczny kurs Java

Wydanie III

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!

- **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 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/nowosci

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

9 788324 632282


Informatyka w najlepszym wydaniu