owoce programowania

Helion

>>> JAVA

DLA ZUPEŁNIE POCZĄTKUJĄCYCH

WYDANIE VII

TONY GADDIS

Tytuł oryginału: Starting Out with Java: From Control Structures through Objects (7th Edition)

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-4829-5

Authorized translation from the English language edition, entitled: STARTING OUT WITH JAVA: FROM CONTROL STRUCTURES THROUGH OBJECTS, Seventh Edition; ISBN 0134802217; by Tony Gaddis; published by Pearson Education, Inc.

Helion SA
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/javzp2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/javzp2.zip

Printed in Poland.

Spis treści

ROZDZIAŁ 2

Podstawy Javy

TEMATYKA

2.1. Części programu w Javie

WYJAŚNIENIE: Program w Javie ma części o określonym przeznaczeniu.

Programy w Javie składają się z różnych części. Pierwszy etap w nauce Javy polega na poznaniu tych części. Zaczniemy od prostego przykładu przedstawionego na listingu 2.1.

WSKAZÓWKA: Pamiętaj, że numery wierszy na listingach nie są częścią programów. Te numery pozwalają wskazywać konkretne wiersze programów.

W rozdziale 1. wspomniano, że nazwy plików z kodem źródłowym w Javie mają rozszerzenie *.java*. Nazwa programu z listingu 2.1 to *Simple.java*. Za pomocą kompilatora Javy ten program można skompilować, wywołując następujące polecenie:

```
javac Simple.java
```

Listing 2.1 Plik Simple.java

```
1  // To prosty program w Javie.
2
3  public class Simple
4  {
5    public static void main(String[] args)
6    {
7      System.out.println("Programowanie to świetna zabawa!");
8    }
9  }
```

Kompilator utworzy wtedy następny plik, *Simple.class*, zawierający przekształcony kod bajtowy Javy. Nowy plik można uruchomić za pomocą następującego polecenia:

```
java Simple
```

WSKAZÓWKA: Pamiętaj, że gdy używasz polecenia `java`, nie powinieneś wpisywać rozszerzenia *.class*.

Poniżej pokazano dane wyjściowe z tego programu. Oto co pojawia się na ekranie po uruchomieniu programu:

Dane wyjściowe programu

```
Programowanie to świetna zabawa!
```

Przeanalizujmy przedstawiony program wiersz po wierszu. Oto instrukcja z wiersza 1.:

```
// To prosty program w Javie.
```

Oprócz dwóch początkowych ukośników ten wiersz wygląda jak zwykłe zdanie. Symbole // oznaczają początek komentarza. Kompilator ignoruje wszystko od podwójnego ukośnika do końca wiersza. To oznacza, że w takim wierszu możesz wpisać cokolwiek, a kompilator nie będzie zgłaszał zastrzeżeń. Choć komentarze nie są wymagane, są bardzo istotne dla programistów. Większość programów jest dużo bardziej skomplikowana niż pokazany przykład, a komentarze pomagają objaśniać kod.

Wiersz 2. jest pusty. Programiści często dodają w programach puste wiersze, aby poprawić czytelność kodu. Oto wiersz 3.:

```
public class Simple
```

Ten wiersz to *nagłówek klasy*. Taki nagłówek oznacza początek *definicji klasy*. Klasy służą między innymi do tworzenia kontenerów na aplikacje. W trakcie lektury tej książki dowiesz się więcej o klasach. Na razie zapamiętaj, że program w Javie musi zawierać przynajmniej jedną definicję klasy. Pokazany wiersz kodu zawiera trzy słowa: `public`, `class` i `Simple`. Przyjrzyjmy się bliżej każdemu z nich.

- `public` to słowo kluczowe Javy. Trzeba je zapisywać małymi literami. Jest to tak zwany *specyfikator dostępu*, kontrolujący sposób dostępu do klasy. Specyfikator `public` oznacza nieograniczony dostęp do klasy. Innymi słowy, taka klasa jest „publicznie otwarta”.

- `class` to także słowo kluczowe Javy (również należy je pisać małymi literami). Oznacza ono początek definicji klasy.
- `Simple` to nazwa klasy. Została ona wymyślona przez programistę. Klasę można też nazwać `Pizza`, `Pies` lub w dowolny inny sposób, jak sobie tego programista zażyczy. Nazwy definiowane przez programistę można pisać za pomocą liter dowolnej wielkości.

W skrócie można stwierdzić, że omawiany wiersz kodu informuje kompilator o tym, iż definiowana jest publicznie dostępna klasa o nazwie `Simple`. Oto dwie inne rzeczy, jakie należy wiedzieć o klasach:

- W pliku możesz utworzyć więcej niż jedną klasę, ale w jednym pliku Javy może znajdować się tylko jedna klasa publiczna (`public class`).
- Jeśli w pliku Javy znajduje się klasa publiczna, jej nazwa musi być identyczna z nazwą pliku (bez rozszerzenia *.java*). Przykładowo, w programie z listingu 2.1 znajduje się klasa publiczna `Simple`, dlatego zapisano ją w pliku *Simple.java*.

UWAGA: W Javie wielkość liter ma znaczenie. To oznacza, że wielkie litery są uznawane za zupełnie inne znaki od ich małych odpowiedników. Słowo `Public` nie jest tym samym co `public`, a `Class` różni się od `class`. Niektóre słowa w programach w Javie muszą być całe zapisane małymi literami. W innych słowach mogą występować kombinacje małych i wielkich liter. Dalej w tym rozdziale znajdziesz listę wszystkich słów kluczowych Javy; trzeba je zapisywać małymi literami.

Wiersz 4. zawiera tylko jeden znak:

```
{
```

Jest to lewy (otwierający) nawias klamrowy, powiązany tu z początkiem definicji klasy. Wszystkie instrukcje będące częścią klasy znajdują się w nawiasie klamrowym. Jeśli spojrzysz na ostatni wiersz programu (wiersz 9.), zobaczysz zamykający nawias klamrowy. Wszystko między tymi dwoma nawiasami jest *ciałem* klasy `Simple`. Poniżej jeszcze raz pokazano kod programu. Tym razem ciało definicji klasy jest wyróżnione ciemniejszym kolorem.

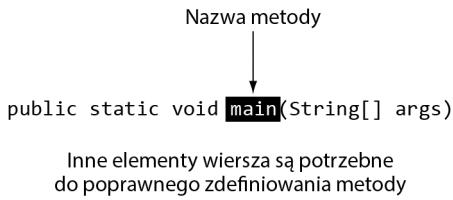
```
// To prosty program w Javie.
public class Simple
{
  public static void main(String[] args)
  {
    System.out.println("Programowanie to świetna zabawa!");
  }
}
```

OSTRZEŻENIE! Upewnij się, że każdemu otwierającemu nawiasowi klamrowemu w programie odpowiada zamykający nawias klamrowy.

Oto wiersz 5.:

```
public static void main(String[] args)
```

Ten wiersz to *nagłówek metody*. Oznacza on początek *metody*. Metodę można traktować jak nazwaną grupę instrukcji programu. Gdy tworzysz metodę, musisz podać kompilatorowi kilka informacji na jej temat. To dlatego przedstawiony wiersz zawiera tyle słów. Na tym etapie ważne jest tylko to, że nazwa metody to `main`. Reszta słów jest potrzebna do poprawnego zdefiniowania tej metody. Nagłówek metody jest pokazany na rysunku 2.1.



Rysunek 2.1. Nagłówek metody main

W rozdziale 1. napisano, że niezależny program w Javie działający w komputerze jest nazywany aplikacją. Każda aplikacja w Javie musi zawierać metodę o nazwie `main`. Ta metoda jest punktem początkowym aplikacji.

UWAGA: Na razie wszystkie programy, jakie będziesz pisał, będą obejmowały klasę z metodą `main`, której nagłówek będzie wyglądał identycznie jak na listingu 2.1. W trakcie lektury tej książki poznasz znaczenie członów `public static void` i `(String[] args)`. Na razie przyjmij, że uczysz się „przepisu" na programy w Javie.

Wiersz 6. zawiera następny otwierający nawias klamrowy:

```
{
```

Ten otwierający nawias klamrowy należy do metody `main`. Pamiętaj, że nawiasy klamrowe otaczają instrukcje, a każdemu nawiasowi otwierającemu musi towarzyszyć nawias zamykający. Jeśli przyjrzysz się wierszowi 8., zobaczysz zamykający nawias klamrowy powiązany z pokazanym tu nawiasem otwierającym. Wszystko między tymi nawiasami to *ciało* metody `main`.

Oto wiersz 7.:

```
System.out.println("Programowanie to świetna zabawa!");
```

Można ująć to prosto — ten wiersz wyświetla na ekranie komunikat. Ten komunikat, „Programowanie to świetna zabawa!", jest wyświetlany bez cudzysłowu. W żargonie programistycznym grupa znaków ujęta w cudzysłów to *literał tekstowy*.

UWAGA: Jest to jedyny wiersz w tym programie, który powoduje wyświetlenie czegoś na ekranie. Inne wiersze, np. `public class Simple` i `public static void main(String[] args)`, są niezbędne do utworzenia szkieletu programu, ale nie wyświetlają niczego na ekranie. Pamiętaj, że program to zestaw instrukcji dla komputera. Jeśli coś ma zostać wyświetlone na ekranie, trzeba posłużyć się w tym celu instrukcją w języku programowania.

Na końcu wiersza znajduje się *średnik*. Podobnie jak kropka oznacza koniec zdania, tak średnik oznacza koniec instrukcji w Javie. Jednak nie każdy wiersz kodu kończy się średnikiem. Poniżej krótko opisano, gdzie nie trzeba umieszczać średnika:

- Komentarze nie muszą kończyć się średnikiem, ponieważ są ignorowane przez kompilator.
- Nagłówki klas i metod nie kończą się średnikiem, ponieważ ich zakończeniem jest blok kodu w nawiasie klamrowym.
- Nawiasy klamrowe { i } nie są instrukcjami, dlatego nie należy umieszczać po nich średnika.

Może się wydawać, że reguły dodawania średników nie są jasne. Na razie skoncentruj się na poznaniu części programu. Wkrótce zrozumiesz, gdzie należy stosować średniki, a gdzie nie są one konieczne.

Jak już wspomniano, wiersze 8. i 9. zawierają zamykające nawiasy klamrowe dla metody `main` i definicji klasy:

```
    }
}
```

Przed przejściem dalej warto przypomnieć omówione kwestie, w tym niektóre mniej oczywiste reguły.

- Java to język, w którym wielkość znaków ma znaczenie. Wielkie litery nie są traktowane w nim tak samo jak ich małe odpowiedniki.
- Wszystkie programy w Javie muszą być zapisywane w plikach o rozszerzeniu *.java*.
- Komentarze są ignorowane przez kompilator.
- Plik *.java* może zawierać wiele klas, ale dozwolona jest tylko jedna klasa publiczna (`public`). Jeśli w pliku *.java* znajduje się klasa publiczna, musi mieć ona tę samą nazwę co sam plik. Przykładowo, jeśli plik *Pizza.java* zawiera klasę publiczną, jej nazwa to `Pizza`.
- Każda aplikacja w Javie musi zawierać metodę o nazwie `main`.
- Dla każdego lewego (otwierającego) nawiasu klamrowego musi istnieć nawias prawy (zamykający).
- Instrukcje kończą się średnikami. Nie dotyczy to komentarzy, nagłówków klas i metod oraz nawiasów klamrowych.

W przykładowym programie znajduje się kilka znaków specjalnych. W tabeli 2.1 znajdziesz podsumowanie ich znaczenia.

Punkt kontrolny

2.1. Ten program nie skompiluje się, ponieważ wymieszano w nim wiersze:

```
public static void main(String[] args)
}
// Zwariowany wymieszany program.
public class Columbus
{
```

Tabela 2.1. Znaki specjalne

Znaki	Nazwa	Znaczenie
//	Podwójny ukośnik	Oznacza początek komentarza.
()	Otwierający i zamykający nawias	Używane w nagłówkach metod.
{}	Otwierający i zamykający nawias klamrowy	Obejmują grupę instrukcji, np. zawartość klasy lub metody.
" "	Cudzysłów	Obejmuje łańcuch znaków, np. komunikat wyświetlany na ekranie.
;	Średnik	Oznacza koniec kompletnej instrukcji języka programowania.

```
System.out.println("W 1492 r. Kolumb wyruszył na błękitny ocean.");
{
}
```

Po właściwym uporządkowaniu wierszy program powinien wyświetlać na ekranie tekst:

```
W 1492 r. Kolumb wyruszył na błękitny ocean.
```

Uporządkuj wiersze we właściwej kolejności. Przetestuj program. W tym celu wpisz go w komputerze, skompiluj i uruchom.

2.2. Jak należy nazwać plik z programem z zadania 2.1?

2.3. Uzupełnij poniższy szkielet programu, aby kod wyświetlał na ekranie tekst „Witaj, świecie!".

```
public class Hello
{
  public static void main(String[] args)
  {
    // Wstaw tu kod, aby uzupełnić program.
  }
}
```

2.4. Zapisz na kartce program, który wyświetli na ekranie Twoje imię. Na początku programu umieść komentarz z dzisiejszą datą. Przetestuj program; w tym celu wpisz go, skompiluj i uruchom.

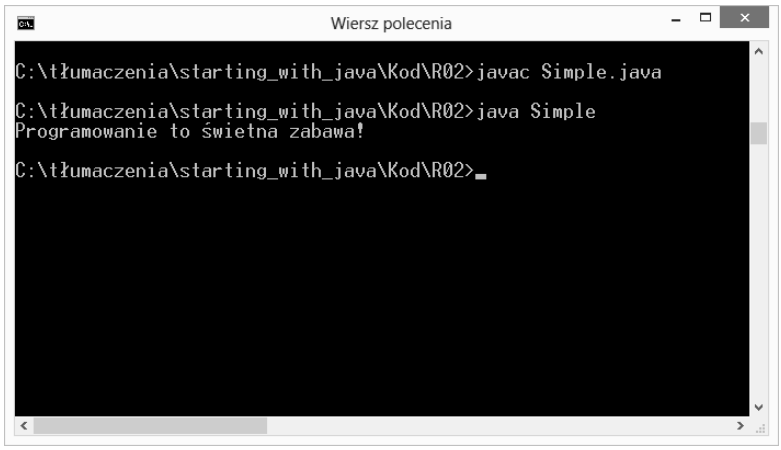
2.5. Nazwy wszystkich plików z kodem źródłowym w Javie muszą się kończyć:
   a. średnikiem,
   b. rozszerzeniem *.class*,
   c. rozszerzeniem *.java*,
   d. żadna z odpowiedzi nie jest poprawna.

2.6. Każda aplikacja w Javie musi zawierać:
   a. metodę o nazwie `main`,
   b. więcej niż jedną definicję klasy,
   c. przynajmniej jeden komentarz.

2.2. Metody print i println oraz interfejs API Javy

WYJAŚNIENIE: Metody `print` i `println` służą do wyświetlania danych tekstowych. Te metody są częścią interfejsu API Javy. Ten interfejs to zestaw gotowych klas i metod do wykonywania określonych operacji.

Z tego podrozdziału dowiesz się, jak pisać programy wyświetlające dane wyjściowe na ekranie. Najprostszy rodzaj danych wyjściowych, jakie program może wyświetlać, to dane w konsoli. *Dane wyjściowe w konsoli* są zwykłym tekstem. Gdy wyświetlasz dane w konsoli w systemie z graficznym interfejsem użytkownika, np. w systemach Windows i macOS, dane wyjściowe zwykle pojawiają się w oknie podobnym do tego z rysunku 2.2.



Rysunek 2.2. Okno konsoli (Microsoft Corporation)

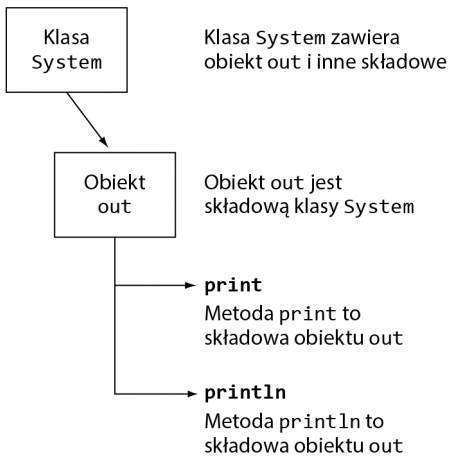
Słowo *konsola* to dawne pojęcie ze świata komputerów. Pochodzi z czasów, gdy operator dużego systemu komputerowego komunikował się z nim, pisząc na terminalu składającym się z prostego ekranu i klawiatury. Ten terminal był nazywany *konsolą*. Ekran konsoli, wyświetlający sam tekst, był nazywany standardowym urządzeniem wyjścia. Obecnie określenie *standardowe urządzenie wyjścia* zwykle oznacza urządzenie wyświetlające dane wyjściowe w konsoli.

W Javie generowanie danych wyjściowych, podobnie jak wiele innych zadań, odbywa się z użyciem interfejsu API Javy. Nazwa *API* to akronim od *Application Programming Interface*. Interfejs API to standardowa biblioteka gotowych klas do wykonywania określonych operacji. Te klasy i ich metody są dostępne we wszystkich programach Javy. Metody `print` i `println` są częścią tego interfejsu API i umożliwiają wyświetlanie danych wyjściowych w standardowym urządzeniu wyjścia.

W programie z listingu 2.1 (*Simple.java*) do wyświetlania komunikatu na ekranie używana jest następująca instrukcja:

```
System.out.println("Programowanie to świetna zabawa!");
```

`System` to klasa z interfejsu API Javy. Ta klasa zawiera obiekty i metody wykonujące operacje na poziomie systemu. Jednym z obiektów z klasy `System` jest `out`. Obiekt `out` zawiera metody (np. `print` i `println`) do wyświetlania danych w konsoli systemowej lub standardowym urządzeniu wyjścia. Hierarchiczne zależności między elementami `System`, `out`, `print` i `println` pokazano na rysunku 2.3.



Rysunek 2.3. Zależności między klasą System, obiektem out a metodami print i println

Oto krótkie podsumowanie współdziałania tych elementów:

- Klasa `System` należy do interfejsu API Javy. Obejmuje składowe obiekty i metody służące do wykonywania operacji na poziomie systemu, np. przesyłania danych wyjściowych do konsoli.
- Obiekt `out` to składowa klasy `System`. Udostępnia metody do przesyłania danych wyjściowych na ekran.
- Metody `print` i `println` to składowe obiektu `out`. Odpowiadają za pokazywanie znaków na ekranie.

Ta hierarchia wyjaśnia, dlaczego instrukcja wykonania metody `println` jest tak długa. W sekwencji `System.out.println` określono, że `println` to składowa obiektu `out`, który jest składową klasy `System`.

Wartość wyświetlana na ekranie jest umieszczona w nawiasie. Ta wartość to *argument*. Przykładowo, następna instrukcja wykonuje metodę `println` z argumentem `"Król Artur"`. Ta instrukcja wyświetla na ekranie tekst „Król Artur” (bez cudzysłowu).

```
System.out.println("Król Artur");
```

Ważne w metodzie `println` jest to, że po wyświetleniu komunikatu przenosi ona kursor na początek następnego wiersza. Kolejny element wyświetlany na ekranie będzie widoczny właśnie tam. Przyjrzyj się programowi z listingu 2.2.

Listing 2.2 Plik TwoLines.java

```
1 // To następny prosty program w Javie.
2
3  public class TwoLines
4  {
5    public static void main(String[] args)
6    {
7      System.out.println("Programowanie to świetna zabawa!");
8      System.out.println("Nie mogę się nim nacieszyć!");
9    }
10 }
```

Dane wyjściowe programu

```
Programowanie to świetna zabawa!
Nie mogę się nim nacieszyć!
```

Ponieważ na listingu 2.2 każdy łańcuch znaków jest wyświetlany za pomocą odrębnej instrukcji `println`, napisy w danych wyjściowych programu pojawiają się w odrębnych wierszach.

Metoda print

Metoda `print`, także należąca do obiektu `System.out`, ma przeznaczenie podobne do metody `println` — wyświetla dane wyjściowe na ekranie. Jednak metoda `print` po wyświetleniu komunikatu nie przenosi kursora do następnego wiersza. Spójrz na kod z listingu 2.3.

Listing 2.3 Plik GreatFun.java

```
1 // To następny prosty program w Javie.
2
3  public class GreatFun
4  {
5    public static void main(String[] args)
6    {
7      System.out.print("Programowanie to ");
8      System.out.println("świetna zabawa!");
9    }
10 }
```

Dane wyjściowe programu

```
Programowanie to świetna zabawa!
```

Ważną kwestią, jaką należy zrozumieć na listingu 2.3, jest to, że choć dane wyjściowe są rozdzielone między dwie instrukcje, program wyświetla komunikat w jednym wierszu. Dane przesyłane do metody `print` są wyświetlane jako ciągły strumień. Czasem prowadzi to do niepożądanych efektów. Przykład przedstawiono na listingu 2.4.

Układ danych wyjściowych różni się od układu łańcuchów znaków z kodu źródłowego. Po pierwsze, choć w kodzie źródłowym dane wyjściowe są rozdzielone na cztery wiersze (od 7. do 10.), na ekranie pojawiają się w jednym wierszu. Po drugie, zauważ, że niektóre

Listing 2.4 Plik Unruly.java

```
1 // Program nieporządnie wyświetlający tekst.
2
3  public class Unruly
4  {
5    public static void main(String[] args)
6    {
7      System.out.print("Oto najlepiej sprzedające się produkty:");
8      System.out.print("Gry komputerowe");
9      System.out.print("Kawa");
10     System.out.println("Aspiryna");
11   }
12 }
```

Dane wyjściowe programu

```
Oto najlepiej sprzedające się produkty:Gry komputeroweKawaAspiryna
```

wyświetlane słowa nie są rozdzielone spacją. Łańcuchy znaków przekazywane do metody `print` są wyświetlane w niezmienionej postaci. Jeśli widoczne mają być spacje, muszą się one znajdować w łańcuchach znaków.

Ten program można naprawić na dwa sposoby. Najbardziej oczywiste jest zastosowanie metod `println` zamiast `print`. Inna możliwość to posłużenie się sekwencjami ucieczki do podziału danych wyjściowych na odrębne wiersze. *Sekwencja ucieczki* rozpoczyna się od lewego ukośnika (\), po którym następują *znaki sterujące*. Za pomocą sekwencji ucieczki można kontrolować sposób wyświetlania danych wyjściowych, umieszczając polecenia w samym łańcuchu znaków. Sekwencja ucieczki powodująca przeniesienie kursora w danych wyjściowych do następnego wiersza to `\n`. Użycie takiej sekwencji ucieczki pokazano na listingu 2.5.

Listing 2.5 Plik Adjusted.java

```
1 // Odpowiednio poprawiony program wyświetlający tekst.
2
3  public class Adjusted
4  {
5    public static void main(String[] args)
6    {
7      System.out.print("Oto najlepiej sprzedające się produkty:\n");
8      System.out.print("Gry komputerowe\nKawa\n");
9      System.out.println("Aspiryna");
10   }
11 }
```

Dane wyjściowe programu

```
Oto najlepiej sprzedające się produkty:
Gry komputerowe
Kawa
Aspiryna
```

Znaki `\n` to sekwencja ucieczki reprezentująca nowy wiersz. Gdy metoda `print` lub `println` natrafia w łańcuchu znaków na sekwencje `\n`, nie wyświetla na ekranie znaków `\n`, ale interpretuje je jako specjalne polecenie przenoszące kursor w danych wyjściowych do następnego wiersza. Istnieją też inne sekwencje ucieczki. Przykła-

dowo, sekwencja ucieczki \t reprezentuje tabulację. Gdy metoda `print` lub `println` napotka w łańcuchu znaków taką sekwencję, przeniesie kursor w danych wyjściowych do następnego punktu tabulacji. Na listingu 2.6 pokazano, jak stosować tę sekwencję.

Listing 2.6 Plik Tabs.java

```
1 // Inny program poprawnie wyświetlający tekst.
2
3  public class Tabs
4  {
5    public static void main(String[] args)
6    {
7      System.out.print("Oto najlepiej sprzedające się produkty:\n");
8      System.out.print("\tGry komputerowe\n\tKawa\n");
9      System.out.println("\tAspiryna");
10   }
11 }
```

Dane wyjściowe programu

```
Oto najlepiej sprzedające się produkty:
   Gry komputerowe
   Kawa
   Aspiryna
```

UWAGA: Choć sekwencja ucieczki składa się z dwóch znaków, w pamięci są one zapisywane jako jeden znak.

W tabeli 2.2 wymieniono i opisano często stosowane sekwencje ucieczki.

Tabela 2.2. Często stosowane sekwencje ucieczki

Sekwencja ucieczki	Nazwa	Opis
\n	Nowy wiersz	Przenosi kursor do następnego wiersza na potrzeby dalszego wyświetlania.
\t	Tabulacja	Przenosi kursor do następnego punktu tabulacji.
\b	Klawisz *Backspace*	Cofa kursor (przenosi go w lewo) o jedną pozycję.
\r	Powrót karetki	Przenosi kursor na początek bieżącego wiersza (nie do następnego wiersza).
\\	Lewy ukośnik	Wyświetla lewy ukośnik.
\'	Apostrof	Wyświetla apostrof.
\"	Cudzysłów	Wyświetla cudzysłów.

OSTRZEŻENIE! Nie pomyl lewego ukośnika (\) z prawym (/). Sekwencja ucieczki nie zadziała, jeśli pomyłkowo rozpoczniesz ją prawym ukośnikiem. Ponadto nie należy umieszczać spacji między lewym ukośnikiem a znakiem sterującym.

Punkt kontrolny

2.7. Ten program nie skompiluje się, ponieważ wymieszano w nim wiersze.

```
System.out.print("Sukces\n");
}
public class Success
{
System.out.print("Sukces\n");
public static void main(String[] args)
System.out.print("Sukces ");
}
// To naprawdę szalony program.
System.out.println("\nSukces");
{
```

Po poprawnym uporządkowaniu wierszy program powinien wyświetlić na ekranie następujące informacje:

Dane wyjściowe programu

```
Sukces
Sukces Sukces

Sukces
```

Uporządkuj wiersze we właściwej kolejności. Przetestuj program. W tym celu zapisz go w komputerze, skompiluj i uruchom.

2.8. Przeanalizuj poniższy program i określ, co wyświetli na ekranie.

```
// Dzieła Wolfganga.
public class Wolfgang
{
  public static void main(String[] args)
  {
    System.out.print("Dzieła Wolfganga\nobejmują ");
    System.out.print("następujące utwory:");
    System.out.print("\nMarsz turecki ");
    System.out.print("i Symfonię nr 40 ");
    System.out.println("g-moll.");
  }
}
```

2.9. Napisz na kartce program, który wyświetli w pierwszym wierszu Twoje imię i nazwisko, w drugim — ulicę oraz numer domu i mieszkania, w trzecim — nazwę miasta i kod pocztowy, a w czwartym — numer telefonu. Na początku programu umieść komentarz z dzisiejszą datą. Przetestuj program; w tym celu wpisz go, skompiluj i uruchom.

2.3. Zmienne i literały

WYJAŚNIENIE: Zmienna to nazwana lokalizacja w pamięci komputera. Literał to wartość zapisana w kodzie programu.

Z rozdziału 1. dowiedziałeś się, że zmienne umożliwiają zapisywanie danych w pamięci komputera i pracę z takimi danymi. Jednym z zadań w trakcie programowania jest

określenie, ilu zmiennych program potrzebuje i jakiego typu dane będą one przechowywać. Na listingu 2.7 pokazano przykładowy program w Javie zawierający zmienną.

Listing 2.7 Plik Variable.java

```
1  // Ten program zawiera zmienną.
2
3  public class Variable
4  {
5    public static void main(String[] args)
6    {
7      int value;
8
9      value = 5;
10     System.out.print("Zmienna value jest równa ");
11     System.out.println(value + ".");
12   }
13 }
```

Dane wyjściowe programu

```
Zmienna value jest równa 5.
```

Przyjrzyjmy się teraz uważnie temu programowi. Oto wiersz 7.:

```
int value;
```

Jest to *deklaracja zmiennej*. Zmienne przed ich użyciem trzeba zadeklarować. Deklaracja zmiennej informuje kompilator o nazwie zmiennej i typie przechowywanych w niej danych. Pokazany wiersz określa, że nazwa zmiennej to `value`. Słowo `int` to skrót od *integer*, czyli „liczba całkowita". Dlatego zmienna `value` może przechowywać tylko liczby całkowite. Warto zauważyć, że deklaracje zmiennych kończą się średnikiem. Następna instrukcja w programie znajduje się w wierszu 9.:

```
value = 5;
```

Jest to *instrukcja przypisania*. Znak równości to operator przypisujący wartość podaną po prawej stronie (tu jest nią 5) do zmiennej podanej po lewej stronie. Po wykonaniu tego wiersza w zmiennej `value` zapisana jest wartość 5.

UWAGA: Ten wiersz nie wyświetla żadnych informacji na ekranie komputera. Działa dyskretnie na zapleczu.

Teraz przyjrzyj się wierszom 10. i 11.:

```
System.out.print("Zmienna value jest równa ");
System.out.println(value + ".");
```

Instrukcja z wiersza 10. przekazuje do metody `print` literał tekstowy `"Zmienna value jest równa "`. Instrukcja z wiersza 11. przekazuje do metody `println` nazwę zmiennej `value`. Przekazanie nazwy zmiennej do metody `print` lub `println` powoduje wyświetlenie wartości tej zmiennej. Zauważ, że nazwa `value` nie jest ujęta w cudzysłów. Zobacz teraz, co się dzieje w programie z listingu 2.8.

Listing 2.8 Plik Variable2.java

```
1  // Ten program zawiera zmienną.
2
3  public class Variable2
4  {
5    public static void main(String[] args)
6    {
7      int value;
8
9      value = 5;
10     System.out.print("Zmienna value jest równa ");
11     System.out.println("value" + ".");
12   }
13 }
```

Dane wyjściowe programu

```
Zmienna value jest równa value.
```

Jeśli słowo `value` jest ujęte w cudzysłów, staje się literałem tekstowym, a nie nazwą zmiennej. Literały tekstowe przesyłane do metod `print` i `println` są wyświetlane w dokładnie takiej postaci, w jakiej występują w cudzysłowach.

Wyświetlanie wielu elementów za pomocą operatora +

Gdy operator + jest używany do łańcuchów znaków, jest nazywany *operatorem konkatenacji (złączania) łańcuchów znaków*. Konkatenacja oznacza łączenie, tak więc operator konkatenacji łączy jeden łańcuch znaków z innym. Przyjrzyj się następującej instrukcji:

```
System.out.println("To jest " + "jeden łańcuch znaków.");
```

Ta instrukcja wyświetla tekst:

```
To jest jeden łańcuch znaków.
```

Operator + tworzy łańcuch znaków będący połączeniem dwóch łańcuchów podanych jako operandy. Operator ten możesz też wykorzystać do złączenia zawartości zmiennej z łańcuchem znaków. Oto przykład:

```
number = 5;
System.out.println("Wartość wynosi " + number + ".");
```

W drugim wierszu zastosowano operator + do złączenia zawartości zmiennej `number` z łańcuchem znaków `"Wartość wynosi "`. Choć `number` nie jest łańcuchem znaków, operator + przekształca wartość tej zmiennej na łańcuch znaków, a następnie złącza tę wartość z pierwszym łańcuchem znaków. Oto wyświetlane dane wyjściowe:

```
Wartość wynosi 5.
```

Czasem argument metody `print` lub `println` jest zbyt długi, aby zmieścić się w jednym wierszu kodu programu. Jednak literał tekstowy nie może zaczynać się w jednym wierszu i kończyć w następnym. Przykładowo, ten kod spowoduje błąd:

```
// To błędny kod!
System.out.println("Wprowadź wartość większą od 0
                    i mniejszą od 10:" );
```

Ten problem można rozwiązać, dzieląc argument na krótsze literały tekstowe i używając operatora złączania łańcuchów znaków. Pozwala to rozdzielić tekst na kilka wierszy. Oto przykład:

```
System.out.println("Wprowadź wartość " +
                   "większą od 0 " +
                   "i mniejszą od 10:" );
```

W tej instrukcji argument jest podzielony na trzy łańcuchy znaków i złączany operatorem +. W następnym przykładzie pokazano tę samą technikę zastosowaną w sytuacji, gdy jednym ze złączanych elementów jest zawartość zmiennej:

```
sum = 249;
System.out.println("Suma trzech liczb " +
                   "wynosi " + sum + ".");
```

Zachowaj ostrożność przy cudzysłowach

Na listingu 2.8 pokazano, że umieszczenie nazwy zmiennej w cudzysłowie zmienia wyniki programu. W rzeczywistości umieszczenie cudzysłowu wokół jakiegokolwiek tekstu, który nie ma być literałem tekstowym, spowoduje jakiegoś rodzaju błąd. Przykładowo, na listingach 2.7 i 2.8 do zmiennej `value` przypisano liczbę 5. To przypisanie wykonane w poniższy sposób jest błędem:

```
value = "5"; // Błąd!
```

W tej instrukcji 5 nie jest liczbą, ale literałem tekstowym. Ponieważ `value` została zadeklarowana jako zmienna całkowitoliczbowa, można w niej zapisywać tylko liczby całkowite. Innymi słowy, `5` i `"5"` oznaczają co innego.

To, że liczby mogą być reprezentowane jako łańcuchy znaków, jest często trudne do zrozumienia dla studentów uczących się programowania. Wystarczy zapamiętać, że łańcuchy znaków są przeznaczone do odczytu przez ludzi. Służą do wyświetlania na ekranach komputerów lub drukowania na papierze. Z kolei liczby są przeznaczone głównie do wykonywania operacji matematycznych. Nie można wykonywać operacji matematycznych na łańcuchach znaków, a liczby przed wyświetleniem na ekranie trzeba przekształcić na łańcuchy znaków. Na szczęście metody `print` i `println` po przekazaniu do nich liczb dokonują takich przekształceń automatycznie. Nie martw się, jeśli wciąż masz wątpliwości. Dalej w rozdziale znajdziesz dokładne omówienie różnic między liczbami, znakami i łańcuchami znaków, związanych ze sposobami ich przechowywania.

Jeszcze o literałach

Literał to wartość zapisana w kodzie programu. Literały są często przypisywane do zmiennych lub wyświetlane. W kodzie listingu 2.9 znajdują się zarówno literały, jak i zmienna.

Zmienną w tym programie jest oczywiście `apples`. Jest ona zadeklarowana jako liczba całkowita. W tabeli 2.3 pokazano listę literałów występujących w tym programie.

Listing 2.9 Plik Literals.java

```
1  // Ten program zawiera literały i zmienną.
2
3  public class Literals
4  {
5    public static void main(String[] args)
6    {
7      int apples;
8
9      apples = 20;
10     System.out.println("Dziś sprzedaliśmy " + apples +
11                        " skrzynek jabłek.");
12   }
13 }
```

Dane wyjściowe programu

```
Dziś sprzedaliśmy 20 skrzynek jabłek.
```

Tabela 2.3. Literały

Literał	Typ literału
20	Literał całkowitoliczbowy
„Dziś sprzedaliśmy ”	Literał tekstowy
„ skrzynek jabłek.”	Literał tekstowy

Identyfikatory

Identyfikator to zdefiniowana przez programistę nazwa reprezentująca jakiś element programu. Nazwy zmiennych i klas to przykładowe identyfikatory. W Javie możesz tworzyć własne nazwy zmiennych i klas, przy czym nie mogą być one słowami kluczowymi tego języka. *Słowa kluczowe* stanowią rdzeń języka, a każde z nich ma określone przeznaczenie. Kompletną listę słów kluczowych Javy znajdziesz w tabeli 1.3 w rozdziale 1.

Zawsze powinieneś stosować takie nazwy zmiennych, aby określały przeznaczenie danej zmiennej. Możliwe, że kuszące będzie dla Ciebie deklarowanie zmiennych o nazwach podobnych do tej:

```
int x;
```

Ta mało opisowa nazwa `x` nie pomaga poznać przeznaczenia zmiennej. Oto lepszy przykład:

```
int posortowaneElementy;
```

Nazwa `posortowaneElementy` pomaga osobom czytającym kod programu zrozumieć, do czego używana jest dana zmienna. Ta metoda pisania kodu pomaga tworzyć *samodokumentujące się programy*, czyli takie, których działanie można zrozumieć na podstawie samej lektury kodu. Ponieważ w praktyce programy mają tysiące wierszy kodu, ważne jest, by były w jak najwyższym stopniu samodokumentujące się.

Prawdopodobnie zwróciłeś uwagę na połączenie wielkich i małych liter w nazwie `posortowaneElementy`. Choć wszystkie słowa kluczowe Javy muszą być zapisywane małymi literami, w nazwach zmiennych można stosować wielkie litery. Litera `E` w nazwie `posortowaneElementy` jest wielka, aby zwiększyć czytelność kodu. Zwykle określenie „posortowane elementy" to dwa słowa. Nazwy zmiennych nie mogą jednak obejmować spacji, dlatego oba słowa trzeba połączyć. Gdy połączysz ze sobą „posortowane" i „elementy", otrzymasz następującą deklarację zmiennej:

```
int posortowaneelementy;
```

Wielka litera `E` sprawia, że łatwiej jest odczytać nazwę `posortowaneElementy`. Nazwy zmiennych zwykle rozpoczynają się małą literą, a dalej początkowe litery każdego kolejnego słowa są wielkie.

Oto reguły, których należy przestrzegać w nazwach wszystkich identyfikatorów:

- Pierwszym znakiem musi być jedna z liter a – z lub A – Z, podkreślenie (_) lub symbol dolara ($).
- Po pierwszym znaku można posługiwać się literami a – z lub A – Z, cyframi 0 – 9, podkreśleniami (_) i symbolem dolara ($).
- Wielkie i małe litery są traktowane jako różne. To oznacza, że nazwa `posortowaneElementy` różni się od nazwy `posortowaneelementy`.
- Identyfikatory nie mogą obejmować spacji.

UWAGA: Choć symbol $ jest dozwolonym znakiem w identyfikatorach, zwykle stosuje się go w specjalnych celach. Dlatego nie należy korzystać z niego w nazwach zmiennych.

W tabeli 2.4 przedstawiono listę nazw zmiennych i wyjaśniono, które z nich są dozwolone w Javie.

Tabela 2.4. Przykładowe nazwy zmiennych

Nazwa zmiennej	Dozwolona czy niedozwolona?
`dzienTygodnia`	Dozwolona.
`3dWykres`	Niedozwolona, ponieważ identyfikatory nie mogą rozpoczynać się cyfrą.
`czerwiec1997`	Dozwolona.
`mieszanka#3`	Niedozwolona, ponieważ w identyfikatorach można stosować tylko litery, cyfry, podkreślenia i symbol dolara.
`dzien tygodnia`	Niedozwolona, ponieważ identyfikatory nie mogą obejmować spacji.

Nazwy klas

Wcześniej wspomniano, że standardową praktyką jest rozpoczynanie nazw zmiennych małą literą i używanie wielkiej litery dla każdego kolejnego słowa w nazwie. Inna standardowa praktyka polega na używaniu wielkiej litery na początku wszystkich słów w nazwach klas. Pomaga to odróżniać nazwy zmiennych od nazw klas. Przykładowo, `payRate` to nazwa zmiennej, a `Employee` to nazwa klasy.

Punkt kontrolny

2.10. Przeanalizuj następujący program:

```
// W tym programie używane są zmienne i literały.
public class BigLittle
{
  public static void main(String[] args)
  {
    int little;
    int big;
    little = 2;
    big = 2000;
    System.out.println("Mała liczba to " + little + ".");
    System.out.println("Duża liczba to " + big + ".");
  }
}
```

Wymień zmienne i literały z tego programu.

2.11. Co poniższy program wyświetli na ekranie?

```
public class CheckPoint
{
  public static void main(String[] args)
  {
    int number;
    number = 712;
    System.out.println("Wartość wynosi " + "number" + ".");
  }
}
```

2.4. Proste typy danych

WYJAŚNIENIE: Istnieje wiele różnych typów danych. Zmienne są klasyfikowane na podstawie ich typu danych, określającego rodzaj danych przechowywanych w zmiennych.

Programy komputerowe pobierają dane z rzeczywistego świata i operują nimi w różny sposób. Istnieje wiele różnych typów danych. Przykładowo, w świecie danych liczbowych istnieją liczby całkowite i ułamkowe, ujemne i dodatnie, a także wartości tak duże i tak małe, że nie mają nazwy. Są też informacje tekstowe. Przykładowo, nazwiska i adresy są zapisywane jako łańcuchy znaków. Gdy piszesz program, musisz ustalić, jakie typy danych prawdopodobnie będą potrzebne.

Każda zmienna ma *typ danych*. Wyznacza on rodzaj danych, jakie zmienna może przechowywać. Wybór właściwego typu danych jest ważny, ponieważ ten typ określa ilość pamięci zajmowanej przez zmienną oraz sposób formatowania i składowania danych. Ważne jest, aby dobrać typ danych odpowiedni dla rodzaju danych używanych w programie. Jeśli piszesz program wyznaczający liczbę mil do odległej gwiazdy, potrzebujesz zmiennych, które potrafią przechowywać bardzo duże wartości. Jeżeli projektujesz oprogramowanie do rejestrowania mikroskopijnych wymiarów, potrzebne będą zmienne przechowujące bardzo małe i precyzyjne liczby. Gdy piszesz program, który musi wykonywać tysiące skomplikowanych obliczeń, przydatne będą zmienne umożliwiające szybkie przetwarzanie. Typ danych zmiennej wpływa na wszystkie te aspekty.

W tabeli 2.5 wymieniono wszystkie *proste typy danych* Javy służące do przechowywania danych liczbowych.

Wyrazy wymienione w lewej kolumnie tabeli 2.5 to słowa kluczowe używane w deklaracjach zmiennych. Deklaracja zmiennych ma następujący ogólny format:

```
TypDanych NazwaZmiennej;
```

Tabela 2.5. Proste liczbowe typy danych

Typ danych	Rozmiar	Zakres
`byte`	1 bajt	Liczby całkowite z przedziału od –128 do +127
`short`	2 bajty	Liczby całkowite z przedziału od –32 768 do +32 767
`int`	4 bajty	Liczby całkowite z przedziału od –2 147 483 648 do +2 147 483 647
`long`	8 bajtów	Liczby całkowite z przedziału od – 9 223 372 036 854 775 808 do +9 223 372 036 854 775 807
`float`	4 bajty	Liczby zmiennoprzecinkowe od $\pm 3{,}4 \times 10^{-38}$ do $\pm 3{,}4 \times 10^{38}$; precyzja na poziomie 7 cyfr
`double`	8 bajtów	Liczby zmiennoprzecinkowe od $\pm 1{,}7 \times 10^{-308}$ do $\pm 1{,}7 \times 10^{308}$; precyzja na poziomie 15 cyfr

`TypDanych` to nazwa typu danych, a *`NazwaZmiennej`* to nazwa zmiennej. Oto kilka przykładowych deklaracji zmiennych:

```
byte inches;
int speed;
short month;
float salesCommission;
double distance;
```

Kolumna *Rozmiar* w tabeli 2.5 określa liczbę bajtów zajmowanych przez zmienne każdego typu danych. Przykładowo, zmienna typu `int` zajmuje 4 bajty, a zmienna typu `double` — 8 bajtów.

W kolumnie *Zakres* określono zakres liczb, jakie można przechowywać w zmiennych poszczególnych typów. Przykładowo, w zmiennej typu `int` można zapisywać liczby od –2 147 483 648 do +2 147 483 647. Jedną z atrakcyjnych cech Javy jest to, że rozmiar i zakres wszystkich prostych typów danych są takie same na każdym komputerze.

UWAGA: Te typy danych są nazywane „prostymi”, ponieważ nie można ich używać do tworzenia obiektów. W rozdziale 1., w omówieniu programowania obiektowego wyjaśniono, że obiekty mają atrybuty i metody. Za pomocą prostych typów danych można jedynie tworzyć zmienne, a w zmiennej można zapisać tylko jedną wartość. Takie zmienne nie mają atrybutów ani metod.

Całkowitoliczbowe typy danych

Cztery pierwsze typy danych wymienione w tabeli 2.5, czyli `byte`, `int`, `short` i `long`, to typy całkowitoliczbowe. Zmienna całkowitoliczbowa może przechowywać liczby całkowite, np. 7, 125, –14 i 6928. W programie z listingu 2.10 używanych jest kilka zmiennych różnych typów całkowitoliczbowych.

Listing 2.10 Plik IntegerVariables.java

```
1  // W tym programie znajdują się zmienne kilku typów całkowitoliczbowych.
2
3  public class IntegerVariables
4  {
5    public static void main(String[] args)
6    {
7      int checking; // Deklaracja zmiennej checking typu int.
8      byte miles; // Deklaracja zmiennej miles typu byte.
9      short minutes; // Deklaracja zmiennej minutes typu short.
10     long days; // Deklaracja zmiennej days typu long.
11
12     checking = -20;
13     miles = 105;
14     minutes = 120;
15     days = 189000;
16     System.out.println("Odbyliśmy podróż o długości " + miles +
17                        " mil.");
18     System.out.println("Zajęło to nam " + minutes + " minut.");
19     System.out.println("Stan naszego konta w złotych wynosi " + checking + ".");
20     System.out.println("Około " + days + " dni temu w tym miejscu " +
21                        "stał Kolumb.");
22   }
23 }
```

Dane wyjściowe programu

```
Odbyliśmy podróż o długości 105 mil.
Zajęło to nam 120 minut.
Stan naszego konta w złotych wynosi –20.
Około 189000 dni temu w tym miejscu stał Kolumb.
```

W większości programów potrzebna jest więcej niż jedna zmienna określonego typu danych. Jeśli w programie używane są trzy liczby całkowite, np. `length`, `width` i `area`, można je zadeklarować osobno:

```
int length;
int width;
int area;
```

Łatwiej jednak jest połączyć deklaracje takich trzech zmiennych:

```
int length, width, area;
```

Można zadeklarować kilka zmiennych tego samego typu, rozdzielając ich nazwy przecinkami.

Literały całkowitoliczbowe

Gdy zapisujesz literał całkowitoliczbowy w kodzie programu, Java przyjmuje, że jest on typu `int`. Przykładowo, na listingu 2.10 literały `–20`, `105`, `120` i `189000` są traktowane jak wartości typu `int`. Możesz jednak wymusić traktowanie takich literałów jak wartości typu `long`. W tym celu należy dodać do liczby literę `L`. Przykładowo, wartość `57L` jest traktowana jak liczba typu `long`. Choć możesz zastosować tu wielką lub małą literę L, zaleca się stosowanie wielkiej litery, ponieważ małe l za bardzo przypomina cyfrę 1.

OSTRZEŻENIE! W literałach liczbowych nie można umieszczać przecinków. Przykładowo, poniższa instrukcja spowoduje błąd:

```
number = 1 , 257 , 649; // BŁĄD!
```

Tę instrukcję należy zapisać w następującej formie:

```
number = 1257649; // Poprawnie.
```

Typy zmiennoprzecinkowe

Liczby całkowite w wielu sytuacjach są niewystarczające. Jeśli piszesz program, który wymaga podawania kwot w złotówkach lub precyzyjnych pomiarów, potrzebujesz typu danych dopuszczającego wartości ułamkowe. W programowaniu są one nazywane liczbami *zmiennoprzecinkowymi*. Wartości takie jak `1.7` i `–45.316` to liczby zmiennoprzecinkowe[1].

W Javie występują dwa typy danych służące do reprezentowania liczb zmiennoprzecinkowych. Te typy to `float` i `double`. Typ `float` to typ danych o pojedynczej precyzji. Pozwala on zapisywać liczby zmiennoprzecinkowe z dokładnością do siedmiu cyfr po przecinku. Typ `double` jest typem danych o podwójnej precyzji. Umożliwia on zapisywanie liczb zmiennoprzecinkowych z dokładnością do 15 cyfr po przecinku. Typ danych `double` zajmuje dwa razy więcej pamięci niż typ danych `float`. Zmienne typu `float` wymagają 4 bajtów pamięci, natomiast zmienne typu `double` — 8 bajtów.

W programie z listingu 2.11 używane są trzy zmienne typu `double`.

[1] Uwaga: w kodzie jako separator części dziesiętnych zawsze używana jest kropka, natomiast przy wyświetlaniu i wprowadzaniu takich wartości zależy to od ustawień komputera, dlatego czasem trzeba zastosować przecinki zamiast kropek — *przyp. tłum.*

Listing 2.11 Plik Sale.java

```
1  // W tym programie pokazano, jak używać typu danych double.
2
3  public class Sale
4  {
5    public static void main(String[] args)
6    {
7      double price, tax, total;
8
9      price = 29.75;
10     tax = 1.76;
11     total = 31.51;
12     System.out.println("Cena produktu " +
13                        "wynosi " + price + ".");
14     System.out.println("Podatek wynosi " + tax + ".");
15     System.out.println("Łączna kwota wynosi " + total + ".");
16   }
17 }
```

Dane wyjściowe programu

```
Cena produktu wynosi 29.75.
Podatek wynosi 1.76.
Łączna kwota wynosi 31.51.
```

Literały zmiennoprzecinkowe

Gdy w kodzie programu umieścisz literał zmiennoprzecinkowy, Java przyjmie, że jest to wartość typu `double`. Na przykład na listingu 2.11 literały `29.75`, `1.76` i `31.51` są traktowane jak wartości typu `double`. Z tego powodu mogą występować problemy w momencie przypisywania literałów zmiennoprzecinkowych do zmiennych typu `float`. Java jest językiem ze *ścisłą kontrolą typów*. To oznacza, że w zmiennych można zapisywać tylko wartości zgodnych typów danych. Wartość `double` nie jest zgodna ze zmienną typu `float`, ponieważ może przyjmować znacznie większe lub znacznie mniejsze wartości, wychodzące poza zakres typu `float`. Dlatego poniższy kod jest błędny:

```
float number;
number = 23.5; // Błąd!
```

Możesz jednak wymusić traktowanie literałów typu `double` jako wartości typu `float`. W tym celu do wartości należy dodać literę `F` lub `f`. Wcześniejszy kod można zmodyfikować w pokazany poniżej sposób, aby uniknąć błędu:

```
float number;
number = 23.5F; // Ten kod zadziała.
```

OSTRZEŻENIE! Jeśli używasz literałów reprezentujących kwotę w dolarach, pamiętaj, że w literałach nie można umieszczać symbolu waluty (np. $) ani przecinków lub spacji. Przykładowo, poniższa instrukcja spowoduje błąd:

```
grossPay = $1,257.00;  // BŁĄD!
```

Tę instrukcję należy zapisać w następujący sposób:

```
grossPay = 1257.00;  // Poprawnie.
```

Notacja naukowa i notacja E

Literały zmiennoprzecinkowe można reprezentować za pomocą notacji naukowej. Przyjrzyj się np. liczbie 47 281,97. W notacji naukowej jej zapis to $4{,}728197 \times 10^4$ (10^4 jest równe 10 000, a $4{,}728197 \times 10\,000$ to 47 281,97).

W Javie do reprezentowania wartości w notacji naukowej używana jest notacja E. W tej notacji liczba $4{,}728197 \times 10^4$ jest zapisywana jako 4,728197E4. W tabeli 2.6 pokazano inne liczby reprezentowane w notacjach naukowej i E.

Tabela 2.6. Reprezentacje liczb zmiennoprzecinkowych

Notacja dziesiętna	Notacja naukowa	Notacja E
247,91	$2{,}4791 \times 10^2$	2,4791E2
0,00072	$7{,}2 \times 10^{-4}$	7,2E – 4
2 900 000	$2{,}9 \times 10^6$	2,9E6

UWAGA: Litera E może być tu wielka lub mała.

Na listingu 2.12 pokazano, jak posługiwać się literałami zmiennoprzecinkowymi w notacji E.

Listing 2.12 Plik SunFacts.java

```
1  // W tym programie używana jest notacja E.
2
3  public class SunFacts
4  {
5    public static void main(String[] args)
6    {
7      double distance, mass;
8
9      distance = 1.495979E11;
10     mass = 1.989E30;
11     System.out.println("Odległość Ziemi od Słońca wynosi " + distance +
12                        " metra.");
13     System.out.println("Masa Słońca to " + mass +
14                        " kilograma.");
15   }
16 }
```

Dane wyjściowe programu

```
Odległość Ziemi od Słońca wynosi 1.495979E11 metra.
Masa Słońca to 1.989E30 kilograma.
```

Typ danych boolean

Typ danych `boolean` umożliwia tworzenie zmiennych, które mogą przyjmować jedną z dwóch wartości: `true` lub `false`. Na listingu 2.13 pokazano deklarowanie zmiennej typu `boolean` i przypisywanie do niej wartości.

Listing 2.13 Plik TrueFalse.java

```
1  // Program przedstawiający używanie zmiennych typu boolean.
2
3  public class TrueFalse
4  {
5    public static void main(String[] args)
6    {
7      boolean bool;
8
9      bool = true;
10     System.out.println(bool);
11     bool = false;
12     System.out.println(bool);
13   }
14 }
```

Dane wyjściowe programu

```
true
false
```

Zmienne typu `boolean` są przydatne do sprawdzania warunków, które mogą być spełnione lub nie. Takie zmienne zaczniesz jednak stosować dopiero w rozdziale 3., dlatego na razie zapamiętaj tylko tyle:

- Zmienne typu `boolean` mogą przyjmować jedynie wartość `true` lub `false`.
- Zawartości zmiennej typu `boolean` nie można skopiować do zmiennej żadnego innego typu.

Typ danych char

Typ danych `char` służy do przechowywania znaków. Zmienna typu `char` może zawierać tylko jeden znak. Literały znakowe są umieszczane w *apostrofach*. W programie z listingu 2.14 używana jest zmienna typu `char`. Przypisywane są do niej literały znakowe `'A'` i `'B'`.

Listing 2.14 Plik Letters.java

```
1  // Ten program ilustruje używanie typu danych char.
2
3  public class Letters
4  {
5    public static void main(String[] args)
6    {
7      char letter;
8
```

```
9      letter = 'A';
10     System.out.println(letter);
11     letter = 'B';
12     System.out.println(letter);
13   }
14 }
```

Dane wyjściowe programu

```
A
B
```

Ważne jest, aby nie mylić literałów znakowych z tekstowymi (te ostatnie są ujmowane w cudzysłów). Literałów tekstowych nie można przypisywać do zmiennych typu `char`.

Unicode

Znaki są wewnętrznie reprezentowane jako liczby. Każdy drukowalny znak, podobnie jak wiele białych znaków, ma przypisaną unikatową liczbę. W Javie używane jest kodowanie Unicode, czyli zestaw liczb stosowanych jako kody reprezentujące znaki. Każda liczba w kodowaniu Unicode wymaga 2 bajtów pamięci. Dlatego zmienne typu `char` zajmują po 2 bajty. Gdy w pamięci umieszczany jest znak, w rzeczywistości zapisywany jest kod liczbowy. Kiedy komputer ma wyświetlić wartość na ekranie, wyświetla znak odpowiadający określonemu kodowi liczbowemu.

Zauważ, że liczba 65 to kod znaku A, liczba 66 to kod znaku B itd. Na listingu 2.15 pokazano, że gdy korzystasz ze znaków, tak naprawdę pracujesz z liczbami.

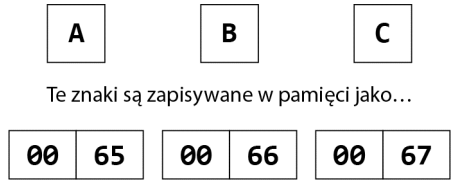
Listing 2.15 Plik Letters2.java

```
1  // Ten program ilustruje ścisłe powiązanie
2  // znaków z liczbami całkowitymi.
3
4  public class Letters2
5  {
6    public static void main(String[] args)
7    {
8      char letter;
9
10     letter = 65;
11     System.out.println(letter);
12     letter = 66;
13     System.out.println(letter);
14   }
15 }
```

Dane wyjściowe programu

```
A
B
```

Na rysunku 2.4 pokazano, że gdy myślisz, że w pamięci zapisywane są znaki A, B i C, w rzeczywistości zachowywane są liczby 65, 66 i 67.



Rysunek 2.4. Znaki i sposób ich zapisywania w pamięci

Inicjowanie zmiennych i przypisywanie do nich wartości

W kilku przykładach pokazano już, że wartość jest umieszczana w zmiennej za pomocą *instrukcji przypisania*. Przykładowo, poniższa instrukcja przypisuje wartość 12 do zmiennej `unitsSold`:

```
unitsSold = 12;
```

Symbol = to operator przypisania. Operatory wykonują operacje na danych. Dane, na których operator działa, to operandy. Operator przypisania ma dwa operandy. W pokazanej tu instrukcji operandami są `unitsSold` i 12.

W instrukcji przypisania nazwa zmiennej, do której przypisywana jest wartość, musi znajdować się po lewej stronie operatora. Przypisywaną wartość należy umieścić po prawej stronie. Poniższa instrukcja jest nieprawidłowa:

```
12 = unitsSold; // BŁĄD!
```

Operandem po lewej stronie operatora = musi być nazwa zmiennej. Operandem po prawej stronie operatora = musi być wyrażenie o jakiejś wartości. Operator przypisania umieszcza wartość prawego operandu w zmiennej określonej przez lewy operand. Przyjmijmy, że `length` i `width` to zmienne typu `int`. Poniższy kod pokazuje, że prawym operandem operatora przypisania mogą być literał lub zmienna:

```
length = 20;
width = length;
```

Warto zauważyć, że operator przypisania zmienia wartość tylko lewego operandu. Druga instrukcja przypisuje wartość zmiennej `length` do zmiennej `width`. Po wykonaniu drugiej instrukcji zmienna `length` zachowa pierwotną wartość, czyli 20.

Wartości można też przypisywać do zmiennej w instrukcji deklaracji. Jest to proces *inicjowania* zmiennej. Pokazano go na listingu 2.16.

Listing 2.16 Plik Initialize.java

```
1  // W tym programie pokazano inicjowanie zmiennych.
2
3  public class Initialize
4  {
5    public static void main(String[] args)
6    {
7      int month = 2, days = 28;
8
9      System.out.println("Miesiąc nr " + month + " ma " +
```

```
10                          days + " dni.");
11    }
12 }
```

Dane wyjściowe programu

```
Miesiąc nr 2 ma 28 dni.
```

Oto instrukcja deklaracji zmiennej z wiersza 7.:

```
int month = 2, days = 28;
```

W tej instrukcji zadeklarowano zmienną `month` i zainicjowano ją wartością 2. Dalej zadeklarowano zmienną `days` i zainicjowano ją wartością 28. Jak widać, upraszcza to program i zmniejsza liczbę instrukcji, jakie programista musi wprowadzić. Oto inne przykładowe deklaracje z inicjowaniem zmiennych:

```
double payRate = 25.52;
float interestRate = 12.9F;
char stockCode = 'D';
int customerNum = 459;
```

Oczywiście istnieją różne odmiany tej techniki. Java umożliwia zadeklarowanie kilku zmiennych i zainicjowanie tylko wybranych z nich. Oto przykład takiej deklaracji:

```
int flightNum = 89, travelTime, departure = 10, distance;
```

Zmienna `flightNum` jest inicjowana wartością 89, a zmienna `departure` — wartością 10. Zmienne `travelTime` i `distance` pozostają niezainicjowane.

OSTRZEŻENIE! Gdy zmienna jest deklarowana w metodzie, najpierw trzeba zapisać wartość w tej zmiennej, a dopiero potem można z niej korzystać. Jeśli kompilator wykryje, że program może użyć zmiennej przed zapisaniem w niej wartości, zgłosi błąd. Błędów tego rodzaju można uniknąć, inicjując zmienną wartością.

Zmienne w danym momencie przechowują tylko jedną wartość

Pamiętaj, że zmienna w danym momencie przechowuje tylko jedną wartość. Gdy przypisujesz do zmiennej nową wartość, zastępuje ona wcześniejszą zawartość zmiennej. Przyjrzyj się np. poniższemu kodowi:

```
int x = 5;
System.out.println(x);
x = 99;
System.out.println(x);
```

W tym kodzie zmienna `x` jest inicjowana wartością 5, po czym zawartość zmiennej jest wyświetlana. Następnie kod przypisuje do zmiennej wartość 99. Zastępuje ona wcześniej zapisaną wartość 5. Ten kod generuje następujące dane wyjściowe:

```
5
99
```

Punkt kontrolny

2.12. Które z poniższych nazw zmiennych są niedozwolone i dlaczego?

```
x
99bottles
july97
theSalesFigureForFiscalYear98
r&d
grade_report
```

2.13. Czy nazwa zmiennej `Sales` jest identyczna z nazwą `sales`? Dlaczego?

2.14. W tym zadaniu posłuż się tabelą 2.5 z prostymi typami danych Javy.
   a) Jeśli zmienna przechowuje liczby całkowite z przedziału od 32 do 6000, to jaki prosty typ danych będzie dla niej najlepszy?
   b) Jeśli zmienna przechowuje liczby całkowite z przedziału od –40 000 do 40 000, to jaki prosty typ danych będzie dla niej najlepszy?
   c) Który z następujących literałów zajmuje więcej pamięci: `22.1` czy `22.1F`?

2.15. Jak liczba $6{,}31 \times 10^{17}$ będzie reprezentowana w notacji E?

2.16. W programie zadeklarowana jest zmienna `number` typu `float`. Poniższa instrukcja powoduje błąd. Jak go naprawić?

```
number = 7.4;
```

2.17. Jakie wartości można przechowywać w zmiennych typu `boolean`?

2.18. Napisz instrukcje wykonujące następujące operacje:
   a) deklarowanie zmiennej `letter` typu `char`,
   b) przypisywanie litery `A` do zmiennej `letter`,
   c) wyświetlanie zawartości zmiennej `letter`.

2.19. Jakie są kody Unicode znaków `'C'`, `'F'` i `'W'`? Możliwe, że będziesz musiał poszukać tych informacji w internecie.

2.20. Który zapis przedstawia literał znakowy: `'B'` czy `"B"`?

2.21. Jaki błąd znajduje się w następującej instrukcji?

```
char letter = "Z";
```

2.5. Operatory arytmetyczne

WYJAŚNIENIE: Jest wiele operatorów służących do manipulowania wartościami liczbowymi i wykonywania operacji arytmetycznych.

Java udostępnia wiele operatorów do manipulowania danymi. Na ogólnym poziomie istnieją trzy rodzaje operatorów: *jednoargumentowe*, *dwuargumentowe* i *trójargumentowe*. Te nazwy określają liczbę operandów wymaganych w poszczególnych operatorach.

Operatory jednoargumentowe wymagają tylko jednego operandu. Przyjrzyj się następującemu wyrażeniu:

```
–5
```

Zrozumiałe jest, że to wyrażenie reprezentuje wartość „minus pięć”. Ten operator można też zastosować do zmiennej:

```
–number
```

To wyrażenie zmienia znak wartości zapisanej w zmiennej `number`. Znak minus używany w ten sposób to *operator negacji*. Ponieważ potrzebny jest tu tylko jeden operand, minus jest operatorem jednoargumentowym.

Operatory dwuargumentowe działają na dwóch operandach. Do tej kategorii należy operator przypisania. Operatory trójargumentowe, jak może się domyśliłeś, wymagają trzech operandów. W Javie dostępny jest tylko jeden operator trójargumentowy; jego opis znajdziesz w rozdziale 3.

Operacje arytmetyczne są w programowaniu bardzo częste. W tabeli 2.7 pokazano operatory arytmetyczne z Javy.

Tabela 2.7. Operatory arytmetyczne

Operator	Znaczenie	Typ	Przykład
`+`	Dodawanie	Dwuargumentowy	`total = cost + tax;`
`–`	Odejmowanie	Dwuargumentowy	`cost = total – tax;`
`*`	Mnożenie	Dwuargumentowy	`tax = cost * rate;`
`/`	Dzielenie	Dwuargumentowy	`salePrice = original / 2;`
`%`	Modulo	Dwuargumentowy	`remainder = value % 3;`

Wszystkie te operatory działają w oczekiwany sposób. Operator dodawania zwraca sumę dwóch operandów. Oto przykładowe instrukcje z wykorzystaniem operatora dodawania:

```
amount = 4 + 8;    // Przypisywanie 12 do zmiennej amount.
total = price + tax;    // Przypisywanie sumy price + tax do zmiennej total.
number = number + 1;    // Przypisywanie sumy number + 1 do zmiennej number.
```

Operator odejmowania zwraca wynik odjęcia prawego operandu od lewego. Oto przykłady:

```
temperature = 112 – 14;    // Przypisywanie 98 do zmiennej temperature.
sale = price – discount;    // Przypisywanie różnicy price – discount do zmiennej sale.
number = number – 1;    // Przypisywanie różnicy number – 1 do zmiennej number.
```

Operator mnożenia zwraca iloczyn dwóch operandów. Oto przykłady:

```
markUp = 12 * 0.25;    // Przypisywanie 3 do zmiennej markUp.
commission = sales * percent;  // Przypisywanie iloczynu sales * percent do zmiennej commission.
population = population * 2;    // Przypisywanie iloczynu population * 2 do zmiennej population.
```

Operator dzielenia zwraca wynik podzielenia lewego operandu przez prawy. Oto przykłady:

```
points = 100 / 20;     // Przypisywanie 5 do zmiennej points.
teams = players / maxEach;    // Przypisywanie ilorazu players / maxEach do zmiennej teams.
half = number / 2;     // Przypisywanie ilorazu number / 2 do zmiennej half.
```

Operator modulo zwraca resztę z operacji dzielenia dwóch liczb całkowitych. Poniższa instrukcja przypisuje liczbę 2 do zmiennej `leftOver`:

```
leftOver = 17 % 3;
```

W niektórych sytuacjach potrzebna jest reszta z dzielenia. Operator modulo jest używany w obliczeniach wykrywających liczby nieparzyste lub wymagających określenia liczby elementów pozostałych po dzieleniu.

Na listingu 2.17 przedstawiono zastosowanie tych operatorów w prostych obliczeniach płac.

Listing 2.17 Plik Wages.java

```
1  // Ten program oblicza wypłatę podstawową plus dodatek za nadgodziny.
2
3  public class Wages
4  {
5    public static void main(String[] args)
6    {
7      double regularWages; // Wypłata podstawowa.
8      double basePay = 25; // Stawka podstawowa.
9      double regularHours = 40; // Przepracowane godziny bez nadgodzin.
10     double overtimeWages; // Wypłata za nadgodziny.
11     double overtimePay = 37.5; // Stawka godzinowa za nadgodziny.
12     double overtimeHours = 10; // Liczba nadgodzin.
13     double totalWages; // Wypłata w sumie.
14
15     regularWages = basePay * regularHours;
16     overtimeWages = overtimePay * overtimeHours;
17     totalWages = regularWages + overtimeWages;
18     System.out.println("Wypłata za ten tydzień wynosi " +
19                        totalWages + "złotych.");
20   }
21 }
```

Dane wyjściowe programu

```
Wypłata za ten tydzień wynosi 1375.0 złotych.
```

Listing 2.17 oblicza łączną tygodniową wypłatę pracownika rozliczanego godzinowo. W komentarzach opisano, że zmienne reprezentują pensję podstawową, podstawową stawkę godzinową, liczbę przepracowanych godzin bez nadgodzin, dodatek za nadgodziny, stawkę godzinową za nadgodziny, przepracowane nadgodziny i łączną wypłatę.

W wierszu 15. program mnoży zmienne `basePay` i `regularHours` oraz zapisuje wynik (1000) w zmiennej `regularWages`:

```
regularWages = basePay * regularHours;
```

W wierszu 16. mnożone są zmienne `overtimePay` i `overtimeHours` oraz zapisywany jest wynik (375) w zmiennej `overtimeWages`:

```
overtimeWages = overtimePay * overtimeHours;
```

W wierszu 17. kod dodaje pensję podstawową do dodatku za nadgodziny i zapisuje wynik (1375) w zmiennej `totalWages`:

```
totalWages = regularWages + overtimeWages;
```

Instrukcja `println` w wierszach 18. i 19. wyświetla na ekranie komunikat informujący o tygodniowej wypłacie.

Dzielenie całkowitoliczbowe

Gdy oba operandy operatora dzielenia są liczbami całkowitymi, operator wykonuje *dzielenie całkowitoliczbowe*. To oznacza, że wynikiem dzielenia także jest liczba całkowita. Reszta jest wtedy pomijana. Przyjrzyj się następującemu kodowi:

```
double number;
number = 5 / 2;
```

Ten kod dzieli 5 przez 2 i przypisuje wynik do zmiennej `number`. Jaka wartość zostanie zapisana w zmiennej `number`? Prawdopodobnie zakładasz, że będzie to 2,5, ponieważ taki wynik dzielenia 5 przez 2 wyświetla kalkulator. Jednak skutek wykonania przedstawionego kodu w Javie jest inny. Ponieważ 5 i 2 są liczbami całkowitymi, część ułamkowa wyniku jest pomijana (*przycinana*). W efekcie do zmiennej `number` przypisywana jest wartość 2.

W tym kodzie nie ma znaczenia, że typem zmiennej `number` jest `double`; odrzucanie części ułamkowej ma miejsce przed przypisaniem. Aby operacja dzielenia zwracała wartość zmiennoprzecinkową, jeden z operandów musi być typu zmiennoprzecinkowego. Przykładowo, wcześniejszy kod można zapisać tak:

```
double number;
number = 5.0 / 2;
```

W tym kodzie wartość `5.0` jest traktowana jak liczba zmiennoprzecinkowa, dlatego operacja dzielenia zwraca liczbę zmiennoprzecinkową. Wynikiem tego dzielenia jest `2.5`.

Pierwszeństwo operatorów

Można pisać wyrażenia matematyczne obejmujące wiele operatorów. Poniższa instrukcja przypisuje sumę operandów `17`, `x`, `21` i `y` do zmiennej `answer`:

```
answer = 17 + x + 21 + y;
```

Jednak niektóre wyrażenia nie są równie proste. Przyjrzyj się następującej instrukcji:

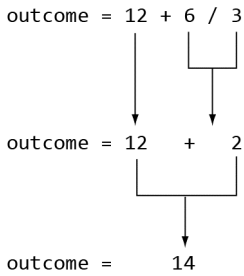
```
outcome = 12 + 6 / 3;
```

Jaka wartość zostanie zapisana w zmiennej `outcome`? Liczba 6 jest tu operandem zarówno operatora dodawania, jak i operatora dzielenia. Do zmiennej `outcome` może zostać przypisana liczba 6 lub 14; zależy to od momentu dzielenia. Rzeczywisty wynik to 14, ponieważ operator dzielenia ma *pierwszeństwo* przed operatorem dodawania.

Wyrażenia matematyczne są przetwarzane od lewej do prawej. Gdy dwa operatory dotyczą tego samego operandu, najpierw wykonywany jest operator o najwyższym pierwszeństwie. Mnożenie i dzielenie mają pierwszeństwo względem dodawania i odejmowania, dlatego przedstawiona instrukcja działa w następujący sposób:

1. 6 jest dzielone przez 3, co daje wynik 2.
2. 12 jest dodawane do 2, co daje wynik 14.

Ten proces pokazano na rysunku 2.5.



Rysunek 2.5. Ilustracja pierwszeństwa

W tabeli 2.8 przedstawiono pierwszeństwo operatorów arytmetycznych. Operatory w górnej części tabeli mają pierwszeństwo przed operatorami położonymi niżej.

Tabela 2.8. Pierwszeństwo operatorów arytmetycznych (od najwyższego do najniższego)

Najwyższe pierwszeństwo	– (jednoargumentowa negacja)
	* / %
Najniższe pierwszeństwo	+ –

Operatory mnożenia, dzielenia i modulo mają ten sam poziom pierwszeństwa. Operatory dodawania i odejmowania też mają to samo pierwszeństwo. Jeśli dwa operatory powiązane z tym samym operandem mają to samo pierwszeństwo, działają zgodnie z zasadą *łączności*. Łączność może być *lewostronna* (operacje wykonywane od lewej do prawej) lub *prawostronna* (operacje wykonywane od prawej do lewej). W tabeli 2.9 wymieniono operatory i ich łączność.

Tabela 2.9. Łączność operatorów arytmetycznych

Operator	Łączność
– (jednoargumentowa negacja)	Prawostronna
* / %	Lewostronna
+ –	Lewostronna

W tabeli 2.10 przedstawiono przykładowe wyrażenia i ich wartości.

Tabela 2.10. Przykładowe wyrażenia i ich wartości

Wyrażenie	Wartość
`5 + 2 * 4`	13
`10 / 2 – 3`	2
`8 + 12 * 2 – 4`	28
`4 + 17 % 2 – 1`	4
`6 – 3 * 2 + 7 – 1`	6

Grupowanie z użyciem nawiasów

Części wyrażeń matematycznych mogą być grupowane za pomocą nawiasów, aby wymusić wykonywanie niektórych operacji przed innymi. W poniższej instrukcji suma zmiennych `a`, `b`, `c` i `d` jest dzielona przez `4.0`:

```
average = (a + b + c + d) / 4.0;
```

Bez nawiasów zmienna `d` zostałaby podzielona przez `4`, a wynik tej operacji zostałby dodany do `a`, `b` i `c`. W tabeli 2.11 pokazano więcej wyrażeń i ich wartości.

Tabela 2.11. Inne wyrażenia i ich wartości

Wyrażenie	Wartość
`(5 + 2) * 4`	28
`10 / (5 – 3)`	5
`8 + 12 * (6 – 2)`	56
`(4 + 17) % (2 – 1)`	0
`(6 – 3) * (2 + 7) / 3`	9

W centrum uwagi:

Obliczanie procentów i rabatów

W programowaniu często trzeba obliczać procenty. Choć w matematyce do określania procentów używany jest symbol %, w większości języków programowania (w tym w Javie) nie stosuje się go w tym celu. W programie trzeba przekształcić procenty na liczbę zmiennoprzecinkową, podobnie jak ma to miejsce w kalkulatorze. Przykładowo, 50% jest zapisywane jako 0,5, a 2% to 0,02.

Przyjrzyj się przykładowi. Załóżmy, że zarabiasz 6000 złotych miesięcznie i część swojej pensji brutto możesz odkładać w ramach planu emerytalnego. Chcesz ustalić, jaką kwotę będziesz odkładał, jeśli na plan emerytalny przeznaczysz 5%, 8% lub 10% pensji brutto. Na potrzeby wspomnianych obliczeń można napisać program taki jak na listingu 2.18.

Listing 2.18 Plik Contribution.java

```
1  // Ten program oblicza kwotę odkładaną w ramach
2  // planu emerytalnego, jeśli przeznaczyć na to 5%,
3  // 8% lub 10% miesięcznego wynagrodzenia.
4
5  public class Contribution
6  {
7    public static void main(String[] args)
8    {
9      // Zmienne przechowujące miesięczne wynagrodzenie
10     // i odkładaną kwotę.
11     double monthlyPay = 6000.0;
12     double contribution;
13
14     // Obliczanie i wyświetlanie kwoty przy odkładaniu 5% pensji.
15     contribution = monthlyPay * 0.05;
16     System.out.println("Odkładanie 5% daje " +
17                        contribution +
18                        " złotych miesięcznie.");
19
20     // Obliczanie i wyświetlanie kwoty przy odkładaniu 8% pensji.
21     contribution = monthlyPay * 0.08;
22     System.out.println("Odkładanie 8% daje " +
23                        contribution +
24                        " złotych miesięcznie.");
25
26     // Obliczanie i wyświetlanie kwoty przy odkładaniu 10% pensji.
27     contribution = monthlyPay * 0.1;
28     System.out.println("Odkładanie 10% daje " +
29                        contribution +
30                        " złotych miesięcznie.");
31   }
32 }
```

Dane wyjściowe programu

```
Odkładanie 5% daje 300.0 złotych miesięcznie.
Odkładanie 8% daje 480.0 złotych miesięcznie.
Odkładanie 10% daje 600.0 złotych miesięcznie.
```

W wierszach 11. i 12. zadeklarowane są dwie zmienne: `monthlyPay` i `contribution`. Zmienna `monthlyPay`, inicjowana wartością `6000.0`, przechowuje kwotę miesięcznego wynagrodzenia. Zmienna `contribution` zawiera kwotę odkładaną w ramach planu emerytalnego.

Instrukcje z wierszy od 15. do 18. obliczają i wyświetlają 5% miesięcznego wynagrodzenia. Obliczenia są wykonywane w wierszu 15., gdzie zmienna `monthlyPay` jest mnożona przez `0.05`. Wynik jest przypisywany do zmiennej `contribution`, wyświetlanej następnie w instrukcji z wierszy od 16. do 18.

Podobne kroki są wykonywane w wierszach od 21. do 24., gdzie obliczana i wyświetlana jest wartość 8% miesięcznej wypłaty, oraz w wierszach od 27. do 30., gdzie obliczane i wyświetlane jest 10% miesięcznego wynagrodzenia.

Obliczanie rabatu o określonym procencie

Inne często wykonywane obliczenia dotyczą rabatu o określonym procencie. Załóżmy, że sklep sprzedaje produkt w pełnej cenie za 59 złotych i planuje urządzić wyprzedaż, w ramach której cena produktu będzie obniżona o 20%. Zostałeś poproszony o napisanie programu obliczającego cenę wyprzedażową.

Aby ustalić cenę wyprzedażową, należy wykonać dwa obliczenia:

- Najpierw ustalić wartość rabatu (20% od pełnej ceny).
- Potem odjąć wartość rabatu od pełnej ceny. W ten sposób uzyskana zostanie cena wyprzedażowa.

Na listingu 2.19 pokazano, jak osiągnąć pożądany efekt w Javie.

Listing 2.19 Plik Discount.java

```
1  // Ten program oblicza cenę wyprzedażową
2  // (po odjęciu rabatu 20%)
3  // produktu o pełnej cenie 59 złotych.
4
5  public class Discount
6  {
7    public static void main(String[] args)
8    {
9      // Zmienne przechowujące pełną cenę, wartość
10     // rabatu i cenę wyprzedażową.
11     double regularPrice = 59.0;
12     double discount;
13     double salePrice;
14
15     // Obliczanie wartości 20% rabatu.
16     discount = regularPrice * 0.2;
17
18     // Obliczanie ceny wyprzedażowej przez odjęcie
19     // rabatu od pełnej ceny.
20     salePrice = regularPrice - discount;
21
22     // Wyświetlanie wyników.
23     System.out.println("Pełna cena: " + regularPrice);
24     System.out.println("Wartość rabatu: " + discount);
25     System.out.println("Cena wyprzedażowa: " + salePrice);
26   }
27 }
```

Dane wyjściowe programu

```
Pełna cena: 59.0
Wartość rabatu: 11.8
Cena wyprzedażowa: 47.2
```

W wierszach od 11. do 13. zadeklarowane są trzy zmienne. Zmienna `regularPrice` przechowuje pełną cenę produktu i jest inicjowana wartością `59.0`. Zmienna `discount` zawiera kwotę rabatu (po jego obliczeniu). W zmiennej `salePrice` zapisana jest cena wyprzedażowa produktu.

W wierszu 16. program oblicza wartość 20% rabatu, mnożąc `regularPrice` przez `0.2`. Wynik jest zapisywany w zmiennej `discount`. W wierszu 20. obliczana jest cena wyprzedażowa. W tym celu od wartości zmiennej `regularPrice` odejmowana jest wartość zmiennej `discount`. Wynik jest zapisywany w zmiennej `salePrice`. Instrukcje od wiersza 23. do wiersza 25. wyświetlają pełną cenę produktu, kwotę rabatu i cenę wyprzedażową.

Klasa Math

Interfejs API Javy udostępnia klasę `Math`, zawierającą liczne metody przydatne w wykonywaniu skomplikowanych operacji matematycznych. W tym punkcie pokrótce omówimy metody `Math.pow` i `Math.sqrt`.

Metoda Math.pow

W Javie podnoszenie liczb do potęgi wymaga użycia metody `Math.pow`. Oto przykład jej zastosowania:

```
result = Math.pow(4.0, 2.0);
```

Ta metoda przyjmuje dwa argumenty typu `double`. Podnosi pierwszy argument do potęgi podanej jako drugi argument i zwraca wynik jako wartość typu `double`. W tym przykładzie wartość `4.0` jest podnoszona do potęgi `2.0`. Ta instrukcja jest odpowiednikiem następującego wyrażenia algebraicznego:

$result = 4^2$

Oto następny przykład instrukcji używającej metody `Math.pow`. Tu wartość 3 razy 6^3 jest przypisywana do zmiennej `x`:

```
x = 3 * Math.pow(6.0, 3.0);
```

Następna instrukcja wyświetla wartość 5 podniesioną do 4. potęgi:

```
System.out.println(Math.pow(5.0, 4.0));
```

Metoda Math.sqrt

Metoda `Math.sqrt` przyjmuje wartość typu `double` i zwraca pierwiastek kwadratowy tej wartości. Oto przykład ilustrujący, jak używać tej metody:

```
result = Math.sqrt(9.0);
```

W tym przykładzie argumentem przekazywanym do metody `Math.sqrt` jest wartość `9.0`. Ta metoda zwraca pierwiastek kwadratowy tej wartości, przypisywany do zmiennej `result`. W kolejnej instrukcji pokazano następny przykład. Tu na ekranie wyświetlany jest pierwiastek kwadratowy wartości `25.0` (czyli `5.0`):

```
System.out.println(Math.sqrt(25.0));
```

Punkt kontrolny

2.22. Uzupełnij poniższą tabelę, wpisując w kolumnie *Wartość* wynik każdego wyrażenia.

Wyrażenie	Wartość
`6 + 3 * 5`	_____
`12 / 2 – 4`	_____
`9 + 14 * 2 – 6`	_____
`5 + 19 % 3 – 1`	_____
`(6 + 2) * 3`	_____
`14 / (11 – 4)`	_____
`9 + 12 * (8 – 3)`	_____

2.23. Czy dzielenie w poniższym kodzie jest całkowitoliczbowe, czy zmiennoprzecinkowe? Jaka wartość zostanie zapisana w zmiennej `portion`?

```
double portion;
portion = 70 / 3;
```

2.6. Złożone operatory przypisania

WYJAŚNIENIE: Złożone operatory przypisania łączą operator przypisania z operatorem arytmetycznym.

W programach często pojawiają się przypisania o następującej postaci:

```
x = x + 1;
```

Po prawej stronie operatora przypisania 1 jest dodawane do `x`. Następnie wynik jest przypisywany do `x`, zastępując wcześniejszą wartość. W efekcie ta instrukcja dodaje 1 do `x`. Oto następny przykład:

```
balance = balance + deposit;
```

Jeśli `balance` i `deposit` to zmienne, ta instrukcja przypisuje wartość wyrażenia `balance + deposit` do zmiennej `balance`. Skutkiem tej instrukcji jest dodanie wartości zmiennej `deposit` do wartości ze zmiennej `balance`. Oto następny przykład:

```
balance = balance - withdrawal;
```

Jeżeli `balance` i `withdrawal` to zmienne, ta instrukcja przypisuje wartość wyrażenia `balance - withdrawal` do zmiennej `balance`. Efektem wykonania tej instrukcji jest odjęcie wartości zmiennej `withdrawal` od wartości zmiennej `balance`.

Jeśli nie zetknąłeś się wcześniej z instrukcjami tego rodzaju, mogą Ci się one początkowo wydawać niezrozumiałe, ponieważ ta sama zmienna znajduje się po obu stronach operatora przypisania. W tabeli 2.12 przedstawiono kolejne instrukcje tego rodzaju.

Tabela 2.12. Różne instrukcje przypisania (w każdej z nich x = 6)

Instrukcja	Działanie instrukcji	Wartość x po wykonaniu instrukcji
`x = x + 4;`	Dodaje 4 do x.	10
`x = x – 3;`	Odejmuje 3 od x.	3
`x = x * 10;`	Mnoży x przez 10.	60
`x = x / 2;`	Dzieli x przez 2.	3
`x = x % 4;`	Przypisuje do x resztę z dzielenia `x / 4`.	2

Operacje tego rodzaju są w programowaniu często wykonywane. Dla wygody programistów Java udostępnia specjalny zestaw operatorów zaprojektowanych specjalnie pod kątem takich operacji. W tabeli 2.13 pokazano *złożone operatory przypisania*, nazywane też operatorami złożonymi.

Tabela 2.13. Złożone operatory przypisania

Operator	Przykładowe zastosowanie	Odpowiednik instrukcji
`+=`	`x += 5;`	`x = x + 5;`
`–=`	`y –= 2;`	`y = y – 2;`
`*=`	`z *= 10;`	`z = z * 10;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`c %= 3;`	`c = c % 3;`

Jak widać, złożone operatory przypisania sprawiają, że programista nie musi dwukrotnie wprowadzać nazwy zmiennej. Poniższą instrukcję:

```
balance = balance + deposit;
```

można przepisać w następujący sposób:

```
balance += deposit;
```

Podobnie instrukcję:

```
balance = balance – withdrawal;
```

można zmodyfikować tak:

```
balance –= withdrawal;
```

Punkt kontrolny

2.24. Napisz z użyciem złożonych operatorów przypisania instrukcje wykonujące następujące operacje:
   a) dodawanie 6 do x,
   b) odejmowanie 4 od `amount`,
   c) mnożenie y przez 4,
   d) dzielenie `total` przez 27,
   e) zapisywanie w x reszty z dzielenia x przez 7.

2.7. Konwersja prostych typów danych na inne takie typy

WYJAŚNIENIE: Aby wartość można było zapisać w zmiennej, typ danych wartości musi być zgodny z typem danych zmiennej. Java automatycznie przeprowadza niektóre konwersje typów danych, jednak nie wykonuje automatycznie żadnych przekształceń, które mogą skutkować utratą danych. Ponadto w Javie stosowany jest zestaw reguł w trakcie przetwarzania wyrażeń arytmetycznych zawierających różne typy danych.

Java to język ze *ścisłą kontrolą typów*. To oznacza, że przed przypisaniem wartości do zmiennej Java sprawdza typy danych zmiennej i wartości, aby ustalić, czy są one zgodne. Przyjrzyj się następującym instrukcjom:

```
int x;
double y = 2.5;
x = y;
```

Instrukcja przypisania próbuje zapisać wartość typu `double` (`2.5`) w zmiennej typu `int`. Gdy kompilator Javy natrafi na ten wiersz kodu, wyświetli komunikat o błędzie. Pakiet JDK wyświetli wtedy komunikat o możliwej utracie precyzji.

Jednak nie wszystkie instrukcje z różnymi typami danych są odrzucane przez kompilator. Przyjrzyj się poniższemu fragmentowi programu:

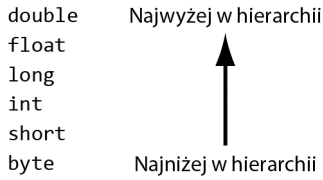
```
int x;
short y = 2;
x = y;
```

Ta instrukcja przypisania, zapisująca wartość typu `short` w zmiennej typu `int`, działa bez problemu. Dlaczego więc Java dopuszcza zapisywanie wartości typu `short` w zmiennych typu `int`, ale nie zezwala na zapis w takich zmiennych wartości typu `double`? Oczywisty powód jest taki, że typ `double` może przechowywać części ułamkowe i wartości znacznie większe niż typ `int`. Gdyby Java dopuszczała zapis wartości typu `double` w zmiennych typu `int`, groziłoby to utratą danych.

Proste typy danych, podobnie jak oficerowie w wojsku, mają określoną hierarchię. Określony typ danych znajduje się wyżej w hierarchii, jeśli może przechowywać większe liczby. Przykładowo, typ `float` znajduje się wyżej w hierarchii niż `int`, a `int` wyżej niż `short`. Na rysunku 2.6 pokazano hierarchię liczbowych typów danych. Im wyżej typ danych znajduje się na tej liście, tym wyższe jest jego miejsce w hierarchii.

W instrukcjach przypisania, gdzie wartości typów danych z niższych poziomów hierarchii są zapisywane w zmiennych z wyższych poziomów, Java automatycznie przekształca wartość na typ z wyższego poziomu hierarchii. Jest to *konwersja poszerzająca zakres*. Taką konwersję przedstawiono w poniższym kodzie, gdzie zachodzi ona w momencie zapisu wartości typu `int` w zmiennej typu `double`:

```
double x;
int y = 10;
x = y;  // Konwersja poszerzająca zakres.
```



Rysunek 2.6. Hierarchia prostych typów danych

Konwersja zawężająca zakres polega na przekształceniu wartości na typ z niższego poziomu hierarchii. Przykładowo, przekształcenie wartości typu `double` na wartość typu `int` to konwersja zawężająca zakres. Ponieważ takie konwersje mogą skutkować utratą danych, Java nie wykonuje ich automatycznie.

Operatory rzutowania

Operator rzutowania umożliwia ręczną konwersję wartości, nawet jeśli oznacza to konwersję zawężającą zakres. Operatory rzutowania są jednoargumentowe i mają postać nazwy typu danych umieszczonej w nawiasie. Taki operator należy podawać przed przekształcaną wartością. Oto przykład:

```
x = (int)number;
```

W tej instrukcji operator rzutowania to słowo `int` w nawiasie. Ten operator zwraca wartość zmiennej `number` przekształconą na typ `int`. Przekształcona wartość jest następnie zapisywana w zmiennej `x`. Jeśli `number` to zmienna zmiennoprzecinkowa, np. typu `float` lub `double`, zwracana wartość jest *przycinana*. Oznacza to, że część ułamkowa liczby zostaje utracona. Pierwotna wartość w zmiennej `number` pozostaje jednak niezmieniona.

W tabeli 2.14 przedstawiono kilka instrukcji z operatorami rzutowania.

Tabela 2.14. Przykładowe zastosowania operatorów rzutowania

Instrukcja	Opis
`littleNum = (short)bigNum;`	Operator rzutowania zwraca tu wartość ze zmiennej `bigNum`, przekształconą na typ `short`. Przekształcona wartość jest przypisywana do zmiennej `littleNum`.
`x = (long)3.7;`	Ten operator rzutowania jest stosowany do wyrażenia `3.7` i zwraca wartość 3, przypisywaną do zmiennej `x`.
`number = (int)72.567;`	Ten operator rzutowania jest stosowany do wyrażenia `72.567` i zwraca wartość 72, używaną do zainicjowania zmiennej `number`.
`value = (float)x;`	Ten operator rzutowania zwraca wartość ze zmiennej `x`, przekształconą na typ `float`. Przekształcona wartość jest przypisywana do zmiennej `value`.
`value = (byte)number;`	Ten operator rzutowania zwraca wartość ze zmiennej `number`, przekształconą na typ `byte`. Przekształcona wartość jest przypisywana do zmiennej `value`.

Warto zauważyć, że zastosowanie operatora rzutowania do zmiennej nie zmienia zawartości tej zmiennej. Operator tylko zwraca zapisaną w zmiennej wartość przekształconą na określony typ danych.

Z wcześniejszej dyskusji warto przypomnieć, że gdy oba operandy dzielenia są liczbami całkowitymi, wykonywane jest dzielenie całkowitoliczbowe. To oznacza, że wynikiem dzielenia będzie liczba całkowita (część ułamkowa wyniku jest pomijana). Przyjrzyj się np. poniższemu kodowi:

```
int pies = 10, people = 4;
double piesPerPerson;
piesPerPerson = pies / people;
```

Choć 10 podzielone przez 4 równa się 2,5, kod zapisuje w zmiennej `piesPerPerson` wartość 2. Ponieważ `pies` i `people` to zmienne typu `int`, wynik też jest typu `int`, a część ułamkowa jest odrzucana. Można jednak zmodyfikować ten kod, używając operatora rzutowania. Dzięki temu uzyskasz poprawny wynik w postaci wartości zmiennoprzecinkowej:

```
piesPerPerson = (double)pies / people;
```

Zmienna `pies` jest typu `int` i zawiera wartość 10. Wyrażenie `(double)pies` zwraca wartość zmiennej `pies` przekształconą na typ `double`. To sprawia, że jeden z operandów operatora dzielenia jest typu `double`, tak więc wynik dzielenia też jest typu `double`. Tę instrukcję można też zapisać w następujący sposób:

```
piesPerPerson = pies / (double)people;
```

W tej instrukcji operator rzutowania zwraca wartość zmiennej `people` przekształconą na typ `double`. W obu instrukcjach wynikiem dzielenia jest wartość typu `double`.

OSTRZEŻENIE! Operator rzutowania można zastosować do całego wyrażenia ujętego w nawias. Przyjrzyj się poniższej przykładowej instrukcji:

```
piesPerPerson = (double)(pies / people);
```

Ta instrukcja nie przekształca wartości ze zmiennych `pies` lub `people` na typ `double`, dokonuje natomiast konwersji wyniku wyrażenia `pies / people`. W tej instrukcji wykonywane jest dzielenie całkowitoliczbowe. Oto wyjaśnienie: wynikiem wyrażenia `pies / people` jest 2 (ponieważ ma miejsce dzielenie całkowitoliczbowe). Wartość 2 przekształcona na typ `double` to `2.0`. Aby zapobiec dzieleniu całkowitoliczbowemu, trzeba przekształcić na typ `double` jeden z operandów.

Operacja na różnych typach całkowitoliczbowych

Jednym z niuansów Javy są wewnętrzne mechanizmy obsługi operacji arytmetycznych na zmiennych typów `int`, `byte` i `short`. Gdy wartości typów `byte` i `short` występują w wyrażeniach arytmetycznych, są tymczasowo przekształcane na wartości typu `int`. Wynik operacji arytmetycznej używającej tylko wartości typów `byte`, `short` lub `int` zawsze jest typu `int`.

Załóżmy np., że b i c w poniższym wyrażeniu są zmiennymi typu short:

```
b + c
```

Choć b i c to zmienne typu short, wynikiem wyrażenia b + c jest wartość typu int. To oznacza, że gdy wynik takiego wyrażenia jest zapisywany w zmiennej, musi być ona typu int lub typu z wyższego poziomu hierarchii. Przyjrzyj się poniższemu kodowi:

```
short firstNumber = 10,
      secondNumber = 20,
      thirdNumber;
// Poniższa instrukcja powoduje błąd!
thirdNumber = firstNumber + secondNumber;
```

W momencie kompilowania tego kodu poniższa instrukcja spowoduje błąd:

```
thirdNumber = firstNumber + secondNumber;
```

Ten błąd wynika z tego, że zmienna thirdNumber jest typu short. Choć firstNumber i secondNumber też są typu short, wynik wyrażenia firstNumber + secondNumber jest typu int. Program można poprawić, deklarując zmienną thirdNumber jako zmienną typu int lub, jak poniżej, stosując operator rzutowania w instrukcji przypisania:

```
thirdNumber = (short)(firstNumber + secondNumber);
```

Inne wyrażenia matematyczne z różnymi typami

W sytuacji gdy wyrażenie matematyczne obejmuje jedną lub kilka wartości typu double, float lub long, Java próbuje przekształcić wszystkie operandy z wyrażenia na ten sam typ danych. Przyjrzyj się regułom rządzącym przetwarzaniem wyrażeń tego rodzaju.

1. Jeśli jeden z operandów operatora jest typu double, wartość drugiego operandu zostanie przekształcona na ten typ. Wynik wyrażenia będzie typu double. Przyjmijmy, że w poniższej instrukcji a jest typu double, a c jest typu int:

   ```
   a = b + c;
   ```

   Wartość c zostanie przed dodawaniem przekształcona na typ double. Wynikiem dodawania będzie wartość typu double, dlatego zmienna a też musi być tego typu.

2. Jeśli jeden z operandów operatora jest typu float, wartość drugiego operandu zostanie przekształcona na ten typ. Wynik wyrażenia będzie typu float. Załóżmy, że w następnej instrukcji x jest typu short, a y jest typu float:

   ```
   z = x * y;
   ```

   Wartość x zostanie przed mnożeniem przekształcona na typ float. Wynikiem mnożenia będzie wartość typu float, dlatego zmienna z musi być typu double lub float.

3. Jeśli jeden z operandów operatora jest typu long, wartość drugiego operandu zostanie przekształcona na ten typ. Wynik wyrażenia będzie typu long. Przyjmijmy, że w poniższej instrukcji a jest typu long, a b jest typu short:

   ```
   c = a - b;
   ```

Zmienna `b` zostanie przed odejmowaniem przekształcona na typ `long`. Wynikiem odejmowania będzie wartość typu `long`, dlatego zmienna `c` musi być typu `long`, `float` lub `double`.

Punkt kontrolny

2.25. W programie znajduje się następująca deklaracja:

```
short totalPay, basePay = 500, bonus = 1000;
```

W tym samym programie występuje taka instrukcja:

```
totalPay = basePay + bonus;
```

a) Czy druga instrukcja zostanie poprawnie skompilowana, czy spowoduje błąd?
b) Jeśli instrukcja spowoduje błąd, to z jakiego powodu tak się stanie? Potrafisz rozwiązać problem?

2.26. Zmienna `a` jest typu `float`, a zmienna `b` jest typu `double`. Napisz instrukcję, która przypisuje wartość `b` do `a`, nie powodując błędu w momencie kompilacji programu.

2.8. Tworzenie nazwanych stałych za pomocą słowa kluczowego final

WYJAŚNIENIE: W deklaracjach zmiennych można zastosować słowo kluczowe `final`, aby utworzyć nazwaną stałą. Nazwane stałe są inicjowane wartością, która nie może się zmieniać w trakcie wykonywania programu.

Załóżmy, że w używanym w banku programie do obliczania danych związanych z kredytami występuje poniższa instrukcja:

```
amount = balance * 0.069;
```

W takim programie powstają dwa problemy. Pierwszy jest taki, że dla osób innych niż autor kodu nie jest jasne, co oznacza wartość `0.069`. Wydaje się, że może to być stopa oprocentowania, jednak czasem z kredytami związane są opłaty. Jak określić znaczenie tej instrukcji bez kłopotliwego sprawdzania reszty programu?

Drugi problem pojawia się, jeśli ta wartość jest używana także w innych obliczeniach w programie i wymaga okresowych modyfikacji. Załóżmy, że ta wartość oznacza stopę oprocentowania. Co się stanie, gdy ta stopa zmieni się z 6,9% na 8,2%? Programista musiałby wtedy przeszukać kod źródłowy pod kątem każdego wystąpienia tej wartości.

Oba te problemy można rozwiązać za pomocą nazwanych stałych. *Nazwana stała* to zmienna, której wartość można tylko odczytać — nie można jej modyfikować w trakcie pracy programu. W Javie taką zmienną można utworzyć za pomocą słowa kluczowego `final` w deklaracji zmiennej. Jest ono umieszczane bezpośrednio przed typem danych. Oto przykład:

```
final double INTEREST_RATE = 0.069;
```

Ta instrukcja wygląda jak zwykła deklaracja zmiennej, przy czym przed typem danych występuje słowo kluczowe `final`, a nazwa zmiennej jest zapisana wielkimi literami. Używanie wielkich liter w nazwie zmiennej nie jest wymagane, jednak wielu programistów preferuje taki zapis, aby móc łatwo odróżniać takie nazwy od nazw zwykłych zmiennych.

Gdy deklarujesz zmienną z modyfikatorem `final`, musisz albo ją zainicjować, albo przypisać do niej wartość. Dopiero potem możesz z niej korzystać. Przykładowo, w poniższym kodzie zmienna o nazwie `PRICE` z modyfikatorem `final` jest deklarowana i inicjowana przed użyciem w operacji wyświetlania danych:

```
final double PRICE = 19.99;
System.out.println("Cena wynosi " + PRICE + ".");
```

W kodzie poniżej jest deklarowana zmienna `PRICE` z modyfikatorem `final`, następnie kod w odrębnej instrukcji przypisuje jej wartość, po czym używa jej w operacji wyświetlania danych:

```
final double PRICE;
PRICE = 19.99;
System.out.println("Cena wynosi " + PRICE + ".");
```

Jednak poniższy kod spowoduje błąd w momencie kompilacji, ponieważ zmiennej z modyfikatorem `final` nie można zastosować przed przypisaniem do niej wartości:

```
final double PRICE;
System.out.println("Cena wynosi " + PRICE + "."); // Błąd!
PRICE = 19.99;
```

Przy przypisaniu wartości do zmiennej z modyfikatorem `final` żadna instrukcja w programie nie może zmienić jej zawartości. Poniższy kod spowoduje błąd kompilacji:

```
final double PRICE;
PRICE = 19.99;
System.out.println("Cena wynosi " + PRICE + ".");
PRICE = 12.99; // Błąd! Nie można zmienić wartości stałej PRICE.
```

Zaletą stosowania nazwanych stałych jest to, że programy są w większym stopniu samodokumentujące się. Poniższą instrukcję:

```
amount = balance * 0.069;
```

można zmodyfikować w następujący sposób:

```
amount = balance * INTEREST_RATE;
```

Nowy programista po przeczytaniu drugiej instrukcji będzie wiedział, do czego ona służy. Oczywiste jest tu, że stan konta jest mnożony przez stopę oprocentowania. Inną zaletą tego podejścia jest możliwość łatwego wprowadzania w programie szeroko zakrojonych zmian. Załóżmy, że stopa oprocentowania pojawia się w kilkunastu różnych instrukcjach programu. Po zmianie stopy wystarczy zmodyfikować wartość inicjującą nazwaną zmienną w jej definicji. Jeśli stopa oprocentowania wzrośnie do 8,2%, deklarację można zmienić w następujący sposób:

```
final double INTEREST_RATE = 0.082;
```

Program jest wtedy gotowy do ponownego skompilowania. W każdej instrukcji, gdzie używana jest stała `INTEREST_RATE`, zastosowana zostanie nowa wartość.

Nazwana stała Math.PI

Klasa `Math`, będąca częścią interfejsu API Javy, udostępnia predefiniowaną nazwaną stałą `Math.PI`. Do tej stałej przypisana jest wartość 3,14159265358979323846, która jest przybliżeniem matematycznej wartości pi. Przyjrzyj się np. poniższej instrukcji:

```
area = Math.PI * radius * radius;
```

Przy założeniu, że zmienna `radius` przechowuje promień okręgu, w tej instrukcji stała `Math.PI` jest używana do obliczania powierzchni okręgu.

2.9. Klasa String

WYJAŚNIENIE: Klasa `String` umożliwia tworzenie obiektów do przechowywania łańcuchów znaków. Obejmuje też różne metody umożliwiające pracę z takimi łańcuchami.

Zetknąłeś się już z łańcuchami znaków i przeanalizowałeś programy wyświetlające je na ekranie. Warto jednak poświęcić chwilę na upewnienie się, czy rozumiesz, czym jest łańcuch znaków. Łańcuch znaków to ciąg znaków. Można go używać do reprezentowania dowolnego rodzaju danych tekstowych, np. nazwisk, adresów, ostrzeżeń itd. Literały tekstowe są umieszczone w cudzysłowie, co pokazano poniżej:

```
"Witaj, świecie!"
"Adam Kowalski"
```

Choć programy często obejmują łańcuchy znaków i muszą wykonywać na nich różne zadania, Java nie udostępnia prostego typu danych do przechowywania takich łańcuchów w pamięci. Zamiast tego interfejs API Javy udostępnia klasę do obsługi łańcuchów znaków. Możesz korzystać z tej klasy do tworzenia obiektów, które potrafią przechowywać łańcuchy znaków i wykonywać na nich operacje. Przed omówieniem tej klasy warto pokrótce wyjaśnić powiązania między klasami a obiektami.

Obiekty są tworzone na podstawie klas

W rozdziale 1. przedstawiono obiekty jako jednostki oprogramowania, które mogą obejmować atrybuty i metody. Atrybuty obiektu to przechowywane w nim dane. Metody obiektu to procedury wykonujące operacje na atrybutach tego obiektu. Przed utworzeniem obiekt musi zostać zaprojektowany przez programistę. Programista określa potrzebne atrybuty i metody, a następnie tworzy klasę opisującą obiekt.

Widziałeś już klasy używane jako kontenery dla aplikacji. Klasa może też posłużyć do określania atrybutów i metod należących do obiektów określonego typu. Klasę możesz traktować jak „wzorzec", na którego podstawie tworzone są obiekty. Klasa nie jest więc obiektem, tylko jego opisem. Gdy program działa, może posłużyć się klasą do utworzenia w pamięci tylu obiektów, ile potrzeba. Każdy obiekt tworzony na podstawie klasy jest nazywany *instancją* tej klasy.

WSKAZÓWKA: Nie martw się, jeśli omawiane zagadnienia wydają Ci się niezrozumiałe. W trakcie lektury tej książki klasy i obiekty będą opisywane wielokrotnie.

Klasa String

Udostępniana w interfejsie API Javy klasa do obsługi łańcuchów znaków to `String`. Pierwszy krok w korzystaniu z tej klasy polega na zadeklarowaniu zmiennej, której typem danych jest klasa `String`. Oto przykład deklaracji zmiennej tego typu:

```
String name;
```

WSKAZÓWKA: W nazwie `String` występuje wielka litera `S`. Zgodnie z konwencją nazwy klas zawsze rozpoczynają się wielką literą.

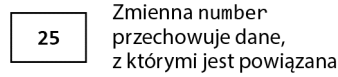
W tej instrukcji zadeklarowano zmienną `name` typu `String`. Pamiętaj, że `String` to klasa, a nie prosty typ danych. Przyjrzyjmy się pokrótce różnicom między zmiennymi typów prostych a zmiennymi, których typem danych jest klasa.

Zmienne typów prostych i zmienne będące instancją klasy

Zmienną dowolnego typu można powiązać z danymi. *Zmienne typów prostych* przechowują konkretne dane, z którymi są powiązane. Załóżmy, że `number` to zmienna typu `int`. Poniższa instrukcja zapisuje w tej zmiennej wartość 25:

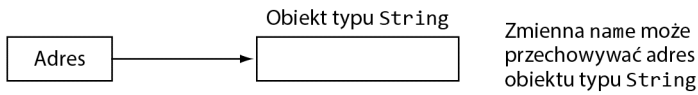
```
number = 25;
```

Ilustruje to rysunek 2.7.



Rysunek 2.7. Zmienna prostego typu danych przechowuje dane, z którymi jest powiązana

Zmienna będąca instancją klasy nie przechowuje danych, z którymi jest powiązana, tylko adres pamięci zajmowany przez te dane. Jeśli `name` jest zmienną będącą instancją klasy `String`, może przechowywać adres obiektu typu `String` zapisanego w pamięci. Przedstawia to rysunek 2.8.



Rysunek 2.8. Zmienna będąca instancją klasy String może przechowywać adres obiektu typu String

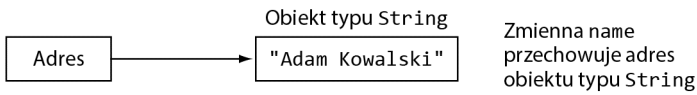
Gdy zmienna będąca instancją klasy przechowuje adres obiektu, zawiera referencję do tego obiektu. Dlatego tego rodzaju zmienne są często nazywane *zmiennymi referencyjnymi*.

Tworzenie obiektu typu String

Gdy zapisujesz literał tekstowy w programie, Java tworzy w pamięci obiekt typu `String` w celu przechowywania tego literału. Aby utworzyć obiekt typu `String` w pamięci i zapisać adres tego obiektu w zmiennej typu `String`, możesz zastosować prostą instrukcję przypisania. Oto przykład:

```
name = "Adam Kowalski";
```

Tu literał tekstowy powoduje utworzenie w pamięci obiektu typu `String` o wartości `"Adam Kowalski"`. Operator przypisania zapisuje adres tego obiektu w zmiennej `name`. Po wykonaniu tej instrukcji zmienna `name` wskazuje obiekt typu `String`. Ilustruje to rysunek 2.9.



Rysunek 2.9. Zmienna name przechowuje adres obiektu typu String

W celu zainicjowania zmiennej typu `String` możesz posłużyć się operatorem =:

```
String name = "Adam Kowalski";
```

Ta instrukcja deklaruje `name` jako zmienną typu `String`, tworzy obiekt typu `String` o wartości `"Adam Kowalski"` i przypisuje adres tego obiektu do zmiennej `name`. Listing 2.20 pokazuje deklarowanie zmiennych typu `String`, ich inicjowanie i używanie w instrukcji `println`.

Listing 2.20 Plik StringDemo.java

```
1  // Prosty program ilustrujący używanie obiektów typu String.
2
3  public class StringDemo
4  {
5    public static void main(String[] args)
6    {
7      String greeting = "Dzień dobry, ";
8      String name = "Henryku.";
9
10     System.out.println(greeting + name);
11   }
12 }
```

Dane wyjściowe programu

```
Dzień dobry, Henryku.
```

Ponieważ typ `String` jest klasą, a nie prostym typem danych, udostępnia liczne metody do pracy z łańcuchami znaków. Klasa `String` ma np. metodę `length`, która zwraca długość łańcucha znaków zapisanego w obiekcie. Jeśli zmienna `name` wskazuje na obiekt typu `String`, poniższa instrukcja zapisuje długość powiązanego z tą zmienną łańcucha znaków w zmiennej `stringSize` (zakładamy tu, że `stringSize` jest zmienną typu `int`):

```
stringSize = name.length();
```

Ta instrukcja wywołuje metodę `length` obiektu, na który wskazuje zmienna `name`. *Wywołanie* metody oznacza jej uruchomienie. Oto ogólna postać wywołania metody:

```
zmiennaWskazująca.metoda(argumenty ...)
```

Tu *`zmiennaWskazująca`* to zmienna wskazująca obiekt, *`metoda`* to nazwa metody, a *`argumenty…`* to zero lub więcej argumentów przekazywanych do tej metody. Jeśli do metody nie są przekazywane żadne argumenty (jak ma to miejsce w metodzie `length`), po nazwie metody trzeba umieścić pusty nawias.

Metoda `length` klasy `String` *zwraca* wartość typu `int`. To oznacza, że metoda przekazuje wartość typu `int` z powrotem do instrukcji, która wywołała tę metodę. Ta wartość może zostać zapisana w zmiennej, wyświetlona na ekranie lub użyta w obliczeniach. Na listingu 2.21 pokazano, jak używać metody `length`.

Listing 2.21 Plik StringLength.java

```
1  // Ten program ilustruje używanie metody length klasy String.
2
3  public class StringLength
4  {
5    public static void main(String[] args)
6    {
7      String name = "Herman";
8      int stringSize;
9
10     stringSize = name.length();
11     System.out.println(name + " składa się z " + stringSize +
12                        " znaków.");
13   }
14 }
```

Dane wyjściowe programu

```
Herman składa się z 6 znaków.
```

UWAGA: Metoda `length` klasy `String` zwraca liczbę znaków w łańcuchu (włącznie ze spacjami).

Z metodami klasy `String` zapoznasz się szczegółowo w rozdziale 9. Na razie przyjrzyj się kilku innym przykładom. Oprócz metody `length` w tabeli 2.15 opisano metody `charAt`, `toLowerCase` i `toUpperCase`.

Korzystanie z tych metod ilustruje program z listingu 2.22.

Tabela 2.15. Kilka metod klasy String

Metoda	Opis i przykład

`charAt(`*`indeks`*`)`	Argument *`indeks`* to wartość typu `int` określająca pozycję znaku w łańcuchu. Pierwszy znak znajduje się na pozycji 0, drugi na pozycji 1 itd. Ta metoda zwraca znak z podanej pozycji. Zwracana wartość jest typu `char`.
	Przykład:
	`char letter;`
	`String name = "Herman";`
	`letter = name.charAt(3);`
	Po wykonaniu tego kodu w zmiennej `letter` znajdzie się znak `'m'`.

`length()`	Ta metoda zwraca liczbę znaków w łańcuchu. Zwracana wartość jest typu `int`.
	Przykład:
	`int stringSize;`
	`String name = "Herman";`
	`stringSize = name.length();`
	Po wykonaniu tego kodu w zmiennej `stringSize` znajdzie się wartość 6.

`toLowerCase()`	Ta metoda zwraca nowy łańcuch znaków. Jest on zapisaną małymi literami wersją łańcucha z obiektu, który wywołał tę metodę.
	Przykład:
	`String bigName = "HERMAN";`
	`String littleName = bigName.toLowerCase();`
	Po wykonaniu tego kodu obiekt wskazywany przez zmienną `littleName` będzie zawierał łańcuch znaków `"herman"`.

`toUpperCase()`	Ta metoda zwraca nowy łańcuch znaków. Jest on zapisaną wielkimi literami wersją łańcucha z obiektu, który wywołał tę metodę.
	Przykład:
	`String littleName = "herman";`
	`String bigName = littleName.toUpperCase();`
	Po wykonaniu tego kodu obiekt wskazywany przez zmienną `bigName` będzie zawierał łańcuch znaków `"HERMAN"`.

Listing 2.22 Plik StringMethods.java

```
1  // Ten program ilustruje używanie kilku metod klasy String.
2
3  public class StringMethods
4  {
5    public static void main(String[] args)
6    {
7      String message = "Java to świetna zabawa!";
8      String upper = message.toUpperCase();
9      String lower = message.toLowerCase();
10     char letter = message.charAt(2);
11     int stringSize = message.length();
12
13     System.out.println(message);
14     System.out.println(upper);
15     System.out.println(lower);
```

```
16     System.out.println(letter);
17     System.out.println(stringSize);
18   }
19 }
```

Dane wyjściowe programu

```
Java to świetna zabawa!
JAVA TO ŚWIETNA ZABAWA!
java to świetna zabawa!
v
18
```

Punkt kontrolny

2.27. Napisz instrukcję, która deklaruje zmienną `city` typu `String`. Tę zmienną należy zainicjować w taki sposób, aby wskazywała obiekt z łańcuchem znaków `"San Francisco"`.

2.28. Załóżmy, że `stringLength` to zmienna typu `int`. Napisz instrukcję, która zapisuje w zmiennej `stringLength` długość łańcucha znaków wskazywanego przez zmienną `city` (zadeklarowaną w punkcie 2.27).

2.29. Przyjmijmy, że `oneChar` to zmienna typu `char`. Napisz instrukcję, która zapisuje w zmiennej `oneChar` pierwszy znak łańcucha wskazywanego przez zmienną `city` (zadeklarowaną w punkcie 2.27).

2.30. Załóżmy, że `upperCity` to zmienna referencyjna typu `String`. Napisz instrukcję, która zapisuje w zmiennej `upperCity` zapisany wielkimi literami odpowiednik łańcucha znaków wskazywanego przez zmienną `city` (zadeklarowaną w punkcie 2.27).

2.31. Załóżmy, że `lowerCity` to zmienna referencyjna typu `String`. Napisz instrukcję, która zapisuje w zmiennej `lowerCity` zapisany mały literami odpowiednik łańcucha znaków wskazywanego przez zmienną `city` (zadeklarowaną w punkcie 2.27).

2.10. Zasięg

WYJAŚNIENIE: Zasięg zmiennej to część programu, która ma dostęp do tej zmiennej.

Każda zmienna ma określony *zasięg*. Zasięg zmiennej to część programu, w której zmienna jest dostępna za pomocą jej nazwy. Zmienna jest widoczna tylko w instrukcjach występujących w zasięgu tej zmiennej. Reguły definiujące zasięg zmiennej są skomplikowane. Tu znajduje się tylko wprowadzenie do tego zagadnienia. W innych rozdziałach temat ten jest opisany dokładniej.

Do tej pory zetknąłeś się jedynie ze zmiennymi deklarowanymi w metodzie `main`. Zmienne deklarowane w metodzie są nazywane *zmiennymi lokalnymi*. Dalej poznasz też zmienne deklarowane poza metodą. Na razie skoncentrujmy się jednak na używaniu zmiennych lokalnych.

Zasięg zmiennej lokalnej rozpoczyna się w miejscu deklaracji zmiennej, a kończy wraz z końcem metody, w której ta zmienna jest zadeklarowana. W instrukcjach poza tym obszarem zmienna jest niedostępna. Oznacza to, że ze zmiennej lokalnej nie można korzystać w kodzie poza metodą, a także w kodzie metody znajdującym się przed deklaracją danej zmiennej. Przykład pokazano na listingu 2.23.

Listing 2.23 Plik Scope.java

```
1  // Ten program nie potrafi znaleźć zmiennej.
2
3  public class Scope
4  {
5    public static void main(String[] args)
6    {
7      System.out.println(value); // BŁĄD!
8      int value = 100;
9    }
10 }
```

Ten program nie skompiluje się, ponieważ próbuje przekazać zawartość zmiennej `value` do metody `println` przed zadeklarowaniem tej zmiennej. Należy pamiętać, że kompilator wczytuje program od początku do końca. Jeśli natrafi na instrukcję, która używa zmiennej przed jej zadeklarowaniem, zgłosi błąd. Aby poprawić program, deklarację zmiennej trzeba umieścić przed korzystającymi z niej instrukcjami.

UWAGA: Jeśli skompilujesz ten program, kompilator wyświetli komunikat o błędzie, np. „cannot resolve symbol" (czyli „nie można zinterpretować symbolu"). To oznacza, że kompilator natrafił na nazwę, której znaczenia nie potrafi określić.

Zgodnie z inną regułą dotyczącą zmiennych lokalnych nie można umieszczać dwóch zmiennych o tej samej nazwie w jednym zasięgu. Przyjrzyj się np. następującej metodzie:

```
public static void main(String[] args)
{
  // Deklarowanie zmiennej number
  // i wyświetlanie jej wartości.
  int number = 7;
  System.out.println(number);
  // Deklarowanie innej zmiennej o nazwie number
  // i wyświetlanie jej wartości.
  int number = 100; // BŁĄD!!!
  System.out.println(number); // BŁĄD!!!
}
```

Ta metoda deklaruje zmienną `number` i inicjuje ją wartością 7. Zasięg tej zmiennej rozpoczyna się wraz z instrukcją deklaracji i rozciąga do końca metody. W zasięgu zmiennej znajduje się instrukcja deklarująca inną zmienną o nazwie `number`. Ta instrukcja powoduje błąd, ponieważ w tym samym zasięgu nie mogą występować dwie zmienne lokalne o tej samej nazwie.

2.11. Komentarze

WYJAŚNIENIE: Komentarze to objaśnienia wierszy dokumentu lub sekcji programu. Komentarze są częścią programu, jednak kompilator je ignoruje; są one przeznaczone dla ludzi, którzy będą czytać kod źródłowy.

Komentarze to krótkie uwagi umieszczane w różnych miejscach programu, wyjaśniające działanie tych części. Komentarze nie są przeznaczone dla kompilatora, tylko dla programistów, którym mają pomagać w zrozumieniu kodu. Kompilator pomija wszystkie komentarze występujące w programie.

Jeśli jesteś początkującym programistą, możliwe, że pomysł zapisywania wielu komentarzy w programach Ci się nie spodoba. W końcu znacznie ciekawsze jest pisanie kodu, który coś robi! Jednak ważne jest, by poświęcać dodatkowy czas na pisanie komentarzy. Niemal na pewno zaoszczędzi Ci to czasu w przyszłości, gdy będziesz musiał modyfikować lub debugować program. Nawet duże i skomplikowane programy mogą być łatwe do czytania i zrozumienia, jeśli są opatrzone odpowiednimi komentarzami.

W Javie występują trzy rodzaje komentarzy: jednowierszowe, wielowierszowe i używane jako dokumentacja. Dalej znajdziesz ich krótkie omówienie.

Komentarze jednowierszowe

Zetknąłeś się już z pierwszym sposobem pisania komentarzy w Javie. Wystarczy umieścić dwa ukośniki (//) w miejscu, gdzie komentarz ma się zaczynać. Kompilator ignoruje wszystko od tego miejsca do końca wiersza. Na listingu 2.24 pokazano, że takie komentarze można swobodnie dodawać w różnych miejscach programu.

Listing 2.24 Plik Comment1.java

```
1  // PROGRAM: Comment1.java
2  // Autor: Herbert Dorfmann
3  // Ten program oblicza płace w firmie.
4
5  public class Comment1
6  {
7    public static void main(String[] args)
8    {
9      double payRate; // Przechowywanie stawki godzinowej.
10     double hours; // Przechowywanie liczby przepracowanych godzin.
11     int employeeNumber; // Przechowywanie numeru pracownika.
12
13     // Resztę programu pominięto.
14   }
15 }
```

Oprócz informowania, kto jest autorem programu, i opisywania przeznaczenia zmiennych komentarze można też wykorzystać do objaśniania skomplikowanych procedur w kodzie.

Komentarze wielowierszowe

Drugi typ komentarzy w Javie to *komentarze wielowierszowe*. Rozpoczynają się one od sekwencji /* (ukośnik, po którym następuje gwiazdka), a kończą sekwencją */ (gwiazdka z ukośnikiem). Wszystko między tymi sekwencjami jest ignorowane. Korzystanie z komentarzy wielowierszowych ilustruje listing 2.25.

Listing 2.25 Plik Comment2.java

```
1  /*
2     PROGRAM: Comment2.java
3     Autor: Herbert Dorfmann
4     Ten program oblicza płace w firmie.
5  */
6
7  public class Comment2
8  {
9     public static void main(String[] args)
10    {
11       double payRate; // Przechowywanie stawki godzinowej.
12       double hours; // Przechowywanie liczby przepracowanych godzin.
13       int employeeNumber; // Przechowywanie numeru pracownika.
14
15       // Resztę programu pominięto.
16    }
17 }
```

Komentarze wielowierszowe (w odróżnieniu od komentarzy rozpoczynających się sekwencją //) mogą rozciągać się na wiele wierszy. Ułatwia to pisanie dużych bloków komentarzy, ponieważ nie trzeba oznaczać każdego wiersza. Komentarze wielowierszowe są z kolei niewygodne do pisania komentarzy jednowierszowych, ponieważ wymagają wpisywania symboli początkowych i końcowych.

Oto wskazówki dotyczące stosowania komentarzy wielowierszowych:

- Uważaj, aby nie zamienić symboli początkowych z końcowymi.
- Uważaj, aby nie zapomnieć o symbolach końcowych.

Wielu programistów stosuje gwiazdki lub inne znaki do rysowania ramek wokół komentarzy. Pomaga to wizualnie oddzielić komentarze od otaczającego je kodu. Powstają w ten sposób komentarze blokowe. Cztery przykładowe komentarze tego typu pokazano w tabeli 2.16.

Tabela 2.16. Komentarze blokowe

```
/*                                  //*************************
 * Ten program ilustruje sposób     // Ten program ilustruje sposób *
 * pisania komentarzy.              // pisania komentarzy.          *
 */                                 //*************************

//////////////////////////          //-------------------------
// Ten program ilustruje sposób     // Ten program ilustruje sposób
// pisania komentarzy.              // pisania komentarzy.
//////////////////////////          //-------------------------
```

Komentarze używane jako dokumentacja

Trzeci rodzaj komentarzy to *komentarze używane jako dokumentacja* (inna nazwa to *komentarze javadoc*). Mogą być one wczytywane i przetwarzane przez program `javadoc`, będący częścią pakietu JDK. Program `javadoc` wczytuje pliki z kodem źródłowym w Javie i generuje atrakcyjnie sformatowane pliki w HTML-u, będące dokumentacją kodu źródłowego. Jeśli pliki z kodem źródłowym zawierają komentarze `javadoc`, informacje z tych komentarzy trafiają do dokumentacji w plikach HTML. Pliki HTML z dokumentacją mogą być wyświetlane w przeglądarce internetowej.

Jako komentarze `javadoc` traktowane są komentarze rozpoczynające się sekwencją `/**` i kończące się sekwencją `*/`. Komentarze `javadoc` zwykle umieszcza się tuż przed nagłówkiem klasy; zawierają wtedy krótki opis danej klasy. Komentarze `javadoc` można też zapisywać przed nagłówkami metod; obejmują wtedy krótkie objaśnienie danej metody. Na listingu 2.26 pokazano program z komentarzami `javadoc`. W tym programie komentarze `javadoc` znajdują się przed nagłówkiem klasy i przed nagłówkiem metody `main`.

Listing 2.26 Plik Comment3.java

```
1  /**
2     Ta klasa tworzy program obliczający pensje w firmie.
3  */
4
5  public class Comment3
6  {
7     /**
8        Metoda main jest punktem wejścia do programu.
9     */
10
11    public static void main(String[] args)
12    {
13       double payRate; // Przechowywanie stawki godzinowej.
14       double hours; // Przechowywanie liczby przepracowanych godzin.
15       int employeeNumber; // Przechowywanie liczby pracowników.
16
17       // Resztę programu pominięto.
18    }
19 }
```

Program `javadoc` można uruchomić w wierszu poleceń systemu operacyjnego. Oto ogólny format wywołania polecenia `javadoc`:

```
javadoc PlikŹródłowy.java
```

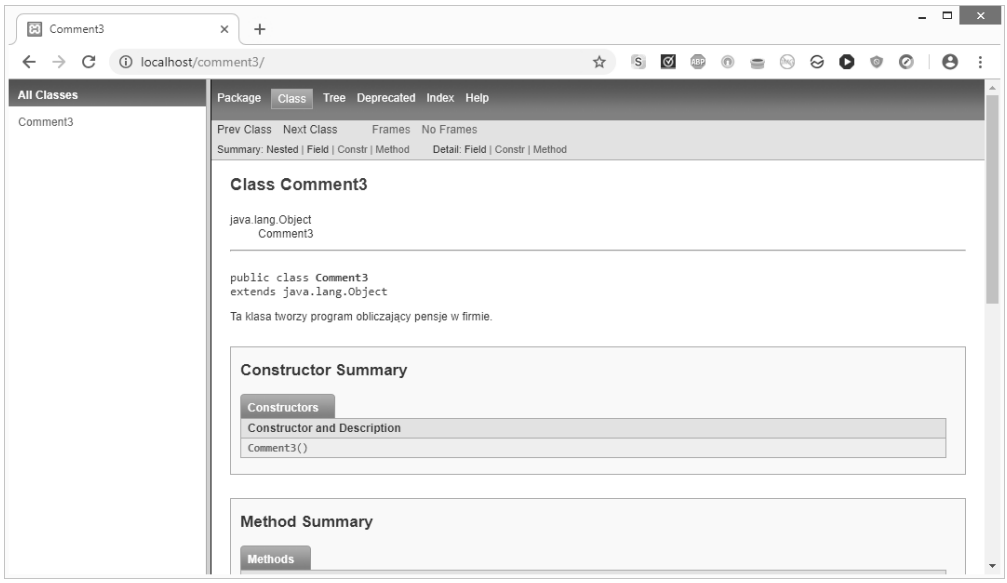
`PlikŹródłowy``.java` to nazwa pliku z kodem źródłowym w Javie (należy pamiętać o dodaniu rozszerzenia `.java`). Program `javadoc` wczytuje ten plik i generuje jego dokumentację. Następna instrukcja utworzy dokumentację pliku z kodem źródłowym *Comment3.java*, pokazanego na listingu 2.26:

```
javadoc Comment3.java
```

Po wykonaniu tego polecenia w katalogu z plikiem z kodem źródłowym utworzonych zostanie kilka plików z dokumentacją. Jeden z nich będzie miał nazwę *index.html*.

Na rysunku 2.10 pokazano plik *index.html* wyświetlony w przeglądarce internetowej. Zauważ, że w nowym pliku znajduje się tekst z komentarzy `javadoc`.

WSKAZÓWKA: Gdy piszesz komentarz `javadoc` dla metody, plik HTML z dokumentacją generowany przez program `javadoc` zawiera dwie sekcje: z podsumowaniem i ze szczegółami. Pierwsze zdanie z komentarza `javadoc` metody jest używane jako podsumowanie. Zauważ, że `javadoc` jako koniec zdania traktuje kropkę ze spacją. Dlatego jeśli opis metody zawiera więcej niż jedno zdanie, zawsze należy zakończyć pierwsze zdanie kropką i spacją. W sekcji ze szczegółami znajduje się cały opis z komentarza `javadoc`.



Rysunek 2.10. Dokumentacja wygenerowana przez program javadoc (Google Inc.)

Jeśli przyjrzysz się dokumentacji pakietu JDK, mającej postać plików HTML wyświetlanych w przeglądarce internetowej, zobaczysz, że jest ona sformatowana w taki sam sposób jak pliki generowane przez program `javadoc`. Zaletą używania programu `javadoc` do dokumentowania kodu źródłowego jest to, że Twoja dokumentacja będzie miała ten sam profesjonalny wygląd i styl co standardowa dokumentacja Javy.

Od tego miejsca książki w przykładowym kodzie źródłowym używane będą komentarze `javadoc`. Wraz z omawianiem różnych zagadnień poznasz dodatkowe zastosowania takich komentarzy i programu `javadoc`.

Punkt kontrolny

2.32. Jak tworzone są komentarze jednowierszowe? Jak należy pisać komentarze wielowierszowe? Jak powstają komentarze `javadoc`?

2.33. Czym komentarze `javadoc` różnią się od innych rodzajów komentarzy?

2.12. Styl programowania

WYJAŚNIENIE: Styl programowania określa sposób, w jaki programista stosuje spacje, wcięcia, puste wiersze i znaki przestankowe do wizualnego porządkowania kodu źródłowego programu.

Z rozdziału 1. dowiedziałeś się, że reguły składni określają sposób posługiwania się językiem. Reguły składni Javy informują, jak i gdzie umieszczać słowa kluczowe, średniki, przecinki, nawiasy klamrowe i inne elementy języka. Kompilator szuka błędów składni, a jeśli program ich nie zawiera, generuje kod bajtowy.

Wczytując program, kompilator przetwarza go jak jeden długi strumień znaków. Dla kompilatora nie ma znaczenia to, że każda instrukcja znajduje się w odrębnym wierszu lub że spacje oddzielają operatory od operandów. Natomiast dla ludzi czytanie programów, które nie są napisane w atrakcyjny wizualnie sposób, jest trudne. Przyjrzyj się np. listingowi 2.27.

Listing 2.27 Plik Compact.java

```
1  public class Compact {public static void main(String [] args){int
2  shares=220; double averagePrice=14.67; System.out.println(
3  "Sprzedano "+shares+" akcji w cenie "+averagePrice+
4  " złotego za akcję.");}}
```

Dane wyjściowe programu

```
Sprzedano 220 akcji w cenie 14.67 złotego za akcję.
```

Choć ten program jest składniowo poprawny (nie narusza żadnych reguł Javy), bardzo trudno się go czyta. Na listingu 2.28 pokazano ten sam program napisany w bardziej czytelnym stylu.

Listing 2.28 Plik Readable.java

```
1  /**
2     Ten przykład jest dużo bardziej czytelny niż plik Compact.java.
3  */
4
5  public class Readable
6  {
7    public static void main(String[] args)
8    {
9      int shares = 220;
10     double averagePrice = 14.67;
11
12     System.out.println("Sprzedano " + shares +
13                        " akcji w cenie " +
14                        averagePrice + " złotego za akcję.");
15}
16}
```

Dane wyjściowe programu

```
Sprzedano 220 akcji w cenie 14.67 złotego za akcję.
```

Nazwa *styl programowania* dotyczy zwykle wizualnego uporządkowania kodu źródłowego. Styl programowania obejmuje techniki spójnego dodawania spacji i wcięć w programie, aby tworzyć wizualne wskazówki. Te wskazówki pozwalają programiście szybko znaleźć ważne informacje na temat programu.

Zwróć uwagę na to, że na listingu 2.28 każdy wiersz w nawiasach klamrowych klasy ma wcięcie, a w metodzie `main` wszystkie wiersze mają jeszcze większe wcięcie. Dodawanie wcięć do wszystkich wierszy w nawiasach klamrowych jest często stosowanym stylem programowania; ilustruje to rysunek 2.11.

```
/**
     Ten przykład jest dużo bardziej czytelny niż plik Compact.java.
*/

public class Readable
{
   public static void main(String[] args)
   {
      int shares = 220;
      double averagePrice = 14.67;

      System.out.println("Sprzedano " + shares +
         " akcji w cenie " +
         averagePrice + " złotego za akcję.");
      }
}
```

Wcięcie, Tu wstawiono dodatkowe spacje

Rysunek 2.11. Wcięcia

Innym aspektem stylu programowania jest zapisywanie instrukcji, które są zbyt długie, aby zmieścić się w jednym wierszu. Zauważ, że instrukcja `println` jest tu rozdzielona na trzy wiersze. Na początku drugiego i trzeciego wiersza instrukcji znajdują się dodatkowe spacje, informujące, że te wiersze są kontynuacją polecenia.

Gdy w jednej instrukcji deklarowanych jest kilka zmiennych tego samego typu, często nazwę każdej zmiennej umieszcza się w odrębnym wierszu wraz z komentarzem objaśniającym przeznaczenie danej zmiennej. Oto przykład:

```
int fahrenheit, // Przechowywanie temperatury w stopniach Fahrenheita.
    celsius,    // Przechowywanie temperatury w stopniach Celsjusza.
    kelvin;     // Przechowywanie temperatury w kelwinach.
```

Możliwe, że zauważyłeś, iż w przykładowych programach między deklaracjami zmiennych a dalszymi instrukcjami znajduje się pusty wiersz. Ma on wizualnie oddzielać wspomniane deklaracje od wykonywalnych instrukcji.

Ze stylem programowania związanych jest też wiele innych kwestii. Są one opisane w różnych miejscach książki.

2.13. Wczytywanie danych wejściowych z klawiatury

WYJAŚNIENIE: Do wczytywania danych wejściowych z klawiatury można wykorzystać obiekty klasy `Scanner`.

Wcześniej opisano obiekt `System.out` i jego związek ze standardowym urządzeniem wyjścia. Interfejs API Javy obejmuje też inny obiekt, `System.in`, wskazujący standardowe urządzenie wejścia. *Standardowym urządzeniem wejścia* jest zwykle klawiatura. Obiekt `System.in` można stosować do wczytywania klawiszy wybranych na klawiaturze. Jednak używanie obiektu `System.in` jest trudniejsze niż korzystanie z obiektu `System.out`. Wynika to z tego, że obiekt `System.in` wczytuje dane wejściowe tylko jako wartości typu `byte`. Przydatność takich danych jest ograniczona, ponieważ programy zwykle potrzebują jako danych wejściowych wartości innych typów. Aby rozwiązać ten problem, można stosować obiekt `System.in` razem z obiektem klasy `Scanner`. Klasa `Scanner` jest zaprojektowana na potrzeby wczytywania danych wejściowych ze źródła (np. obiektu `System.in`) i udostępnia metody, które można wykorzystać do pobierania danych wejściowych sformatowanych jako wartości typów prostych lub łańcuchy znaków.

Najpierw należy utworzyć obiekt typu `Scanner` i powiązać go z obiektem `System.in`. Oto przykładowa instrukcja, która wykonuje tę operację:

```
Scanner keyboard = new Scanner(System.in);
```

Podzielmy tę instrukcję na dwie części. W pierwszej części:

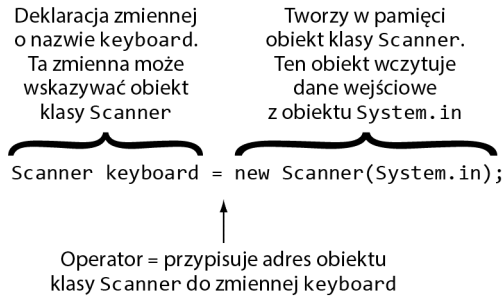
```
Scanner keyboard
```

deklarowana jest zmienna `keyboard`. Typ danych tej zmiennej to `Scanner`. Ponieważ `Scanner` jest klasą, `keyboard` jest zmienną, której typ danych to klasa. W omówieniu obiektów typu `String` napisano, że takie zmienne przechowują adres do pamięci, gdzie zapisany jest obiekt. Dlatego zmienna `keyboard` posłuży do przechowywania adresu obiektu typu `Scanner`. Oto druga część instrukcji:

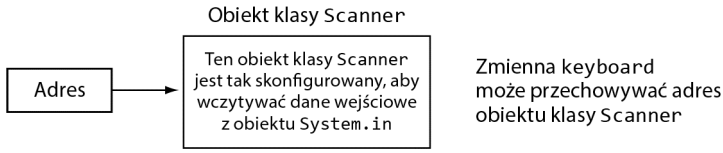
```
= new Scanner(System.in);
```

Pierwszą rzeczą, jaką widać w tej części instrukcji, jest operator przypisania (`=`). Ten operator przypisuje jakieś dane do zmiennej `keyboard`. Po operatorze przypisania znajduje się słowo `new` (jest to słowo kluczowe Javy). Służy ono do tworzenia obiektów w pamięci. Dalej podany jest typ tworzonego obiektu. Tu po słowie kluczowym `new` znajduje się instrukcja `Scanner(System.in)`. Określa ona, że należy utworzyć obiekt typu `Scanner`, który ma być powiązany z obiektem `System.in`. Adres tego obiektu w pamięci jest przypisywany (za pomocą operatora `=`) do zmiennej `keyboard`. Po wykonaniu tej instrukcji zmienna `keyboard` wskazuje utworzony w pamięci obiekt typu `Scanner`.

Na rysunku 2.12 opisane jest przeznaczenie każdego fragmentu tej instrukcji. Rysunek 2.13 pokazuje, że zmienna `keyboard` wskazuje obiekt klasy `Scanner`.



Rysunek 2.12. Części instrukcji



Rysunek 2.13. Zmienna keyboard wskazuje obiekt klasy Scanner

UWAGA: W tym kodzie nazwą zmiennej jest `keyboard` (czyli klawiatura). W tej nazwie nie ma nic wyjątkowego. Wybrano ją, ponieważ zmienna ta posłuży do wczytywania danych wejściowych z klawiatury.

Klasa `Scanner` obejmuje metody do wczytywania łańcuchów znaków, wartości typu `byte`, liczb całkowitych typu `int`, `long` i `short` oraz wartości typu `float` i `double`. Przykładowo, w kodzie poniżej obiekt klasy `Scanner` służy do wczytywania z klawiatury wartości typu `int` i przypisywania tej wartości do zmiennej typu `number`.

```
int number;
Scanner keyboard = new Scanner(System.in);
System.out.print("Wprowadź liczbę całkowitą: ");
number = keyboard.nextInt();
```

Ostatnia pokazana tu instrukcja wywołuje metodę `nextInt` klasy `Scanner`. Ta metoda formatuje wprowadzone dane jako wartość typu `int`, a następnie ją zwraca. Tak więc pokazana instrukcja formatuje wpisane za pomocą klawiatury dane wejściowe jako wartość typu `int`, po czym ją zwraca. Ta wartość jest przypisywana do zmiennej `number`.

W tabeli 2.17 wymieniono kilka metod klasy `Scanner` i opisano ich zastosowanie.

Używanie instrukcji import

Jest jeszcze jeden związany z klasą `Scanner` szczegół, o którym musisz wiedzieć, zanim zaczniesz jej używać. Klasa ta nie jest automatycznie dostępna w programach w Javie. Każdy program, w którym używana jest klasa `Scanner`, musi zawierać w początkowej części pliku (przed definicjami klas) następującą instrukcję:

Tabela 2.17. Wybrane metody klasy Scanner

Metoda	Przykład i opis

`nextByte`	**Przykład zastosowania:**
	`byte x;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu byte: ");`
	`x = keyboard.nextByte();`
	Opis: Zwraca dane wejściowe jako wartość typu `byte`.

`nextDouble`	**Przykład zastosowania:**
	`double number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu double: ");`
	`number = keyboard.nextDouble();`
	Opis: Zwraca dane wejściowe jako wartość typu `double`.

`nextFloat`	**Przykład zastosowania:**
	`float number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu float: ");`
	`number = keyboard.nextFloat();`
	Opis: Zwraca dane wejściowe jako wartość typu `float`.

`nextInt`	**Przykład zastosowania:**
	`int number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu int: ");`
	`number = keyboard.nextInt();`
	Opis: Zwraca dane wejściowe jako wartość typu `int`.

`nextLine`	**Przykład zastosowania:**
	`String name;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź swoje imię: ");`
	`name = keyboard.nextLine();`
	Opis: Zwraca dane wejściowe jako wartość typu `String`.

`nextLong`	**Przykład zastosowania:**
	`long number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu long: ");`
	`number = keyboard.nextLong();`
	Opis: Zwraca dane wejściowe jako wartość typu `long`.

`nextShort`	**Przykład zastosowania:**
	`short number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Wprowadź wartość typu short: ");`
	`number = keyboard.nextShort();`
	Opis: Zwraca dane wejściowe jako wartość typu `short`.

```
import java.util.Scanner;
```

Ta instrukcja informuje kompilator Javy, gdzie w bibliotece Javy znajduje się klasa `Scanner`, oraz udostępnia tę klasę w programie.

Na listingu 2.29 pokazano, jak używać klasy `Scanner` do wczytywania wartości typów `String`, `int` i `double`.

Listing 2.29 Plik Payroll.java

```
1  import java.util.Scanner; // Potrzebne, by móc używać klasy Scanner.
2
3  /**
4     Ten program ilustruje stosowanie klasy Scanner.
5  */
6
7  public class Payroll
8  {
9     public static void main(String[] args)
10    {
11       String name; // Przechowywanie nazwiska.
12       int hours; // Przepracowane godziny.
13       double payRate; // Stawka godzinowa.
14       double grossPay; // Pensja brutto.
15
16       // Tworzenie obiektu typu Scanner do wczytywania danych wejściowych.
17       Scanner keyboard = new Scanner(System.in);
18
19       // Pobieranie nazwiska użytkownika.
20       System.out.print("Jak się nazywasz? ");
21       name = keyboard.nextLine();
22
23       // Pobieranie liczby godzin przepracowanych w tym tygodniu.
24       System.out.print("Ile godzin przepracowałeś w tym tygodniu? ");
25       hours = keyboard.nextInt();
26
27       // Pobieranie stawki godzinowej użytkownika.
28       System.out.print("Ile wynosi stawka godzinowa? ");
29       payRate = keyboard.nextDouble();
30
31       // Obliczanie pensji brutto.
32       grossPay = hours * payRate;
33
34       // Wyświetlanie wynikowych informacji.
35       System.out.println("Witaj, " + name + ".");
36       System.out.println("Twoja pensja brutto wynosi " + grossPay + "złotych.");
37    }
38 }
```

Dane wyjściowe programu (przykładowe dane wejściowe są wyróżnione pogrubieniem)

```
Jak się nazywasz? Adam Kowalski [Enter]
Ile godzin przepracowałeś w tym tygodniu? 40 [Enter]
Ile wynosi stawka godzinowa? 20 [Enter]
Witaj, Adam Kowalski.
Twoja pensja brutto wynosi 800.00 złotych.
```

UWAGA: Zauważ, że każda używana tu metoda klasy `Scanner` oczekuje na wciśnięcie przez użytkownika klawisza *Enter* i dopiero po tym zwraca wartość. Po wciśnięciu klawisza *Enter* kursor jest automatycznie przenoszony do następnego wiersza, gdzie wykonywane są dalsze operacje zwracania danych wyjściowych.

Wczytywanie znaków

Czasem chcesz wczytać jeden znak z klawiatury. Przykładowo, program może zadawać pytania z odpowiedziami tak/nie i informować, że należy wprowadzić T (oznaczające „tak") lub N (oznaczające „nie"). Klasa `Scanner` nie udostępnia jednak metody do wczytywania pojedynczych znaków. W tej książce zastosowano następujący sposób wczytywania znaków: używana jest metoda `nextLine` klasy `Scanner` do wczytania łańcucha znaków z klawiatury, a następnie metoda `charAt` klasy `String` do pobrania pierwszego znaku tego łańcucha. Pobierany jest wtedy znak wprowadzony przez użytkownika za pomocą klawiatury. Oto przykład:

```
String input;  // Przechowywanie wiersza danych wejściowych.
char answer;   // Przechowywanie jednego znaku.

// Tworzenie obiektu typu Scanner do pobierania danych wejściowych z klawiatury.
Scanner keyboard = new Scanner(System.in);

// Zadawanie pytania użytkownikowi.
System.out.print("Czy dobrze się bawisz? (T=tak, N=nie) ");
input = keyboard.nextLine();  // Pobieranie wiersza danych wejściowych.
answer = input.charAt(0);     // Pobieranie pierwszego znaku.
```

Zmienna `input` wskazuje obiekt typu `String`. Ostatnia instrukcja w tym kodzie wywołuje metodę `charAt` klasy `String`, aby pobrać znak z pozycji 0; jest to pierwszy znak z łańcucha. Po wykonaniu tej instrukcji w zmiennej `answer` znajdzie się znak wprowadzony przez użytkownika za pomocą klawiatury.

Łączenie wywołań metody nextLine z wywołaniami innych metod klasy Scanner

Gdy wywołasz jedną z metod klasy `Scanner` służących do wczytywania wartości typu prostego (np. metodę `nextInt` lub `nextDouble`), a następnie wywołasz metodę `nextLine`, aby wczytać łańcuch znaków, może wystąpić irytujący i trudny do wykrycia problem. Spójrz np. na program z listingu 2.30.

Listing 2.30 Plik InputProblem.java

```
1  import java.util.Scanner; // Potrzebne, by móc używać klasy Scanner.
2
3  /**
4     W tym programie występuje problem z wczytywaniem danych wejściowych.
5  */
6
7  public class InputProblem
8  {
9    public static void main(String[] args)
10   {
11     String name; // Przechowywanie nazwy użytkownika.
12     int age; // Przechowywanie wieku użytkownika.
13     double income; // Przechowywanie dochodu użytkownika.
14
15     // Tworzenie obiektu klasy Scanner do wczytywania danych wejściowych.
16     Scanner keyboard = new Scanner(System.in);
```

```
17
18      // Pobieranie wieku użytkownika.
19      System.out.print("Ile masz lat? ");
20      age = keyboard.nextInt();
21
22      // Pobieranie dochodu użytkownika.
23      System.out.print("Ile wynosi Twój roczny dochód? ");
24      income = keyboard.nextDouble();
25
26      // Pobieranie nazwiska użytkownika.
27      System.out.print("Jak się nazywasz? ");
28      name = keyboard.nextLine();
29
30      // Wyświetlanie informacji użytkownikowi.
31      System.out.println("Witaj, " + name + ". Twój wiek to " +
32                         age + ", a Twój dochód wynosi " +
33                         income + " złotych.");
34    }
35 }
```

Dane wyjściowe programu (przykładowe dane wejściowe są wyróżnione pogrubieniem)

```
Ile masz lat? 24 [Enter]
Ile wynosi Twój roczny dochód? 50000.00 [Enter]
Jak się nazywasz? Witaj, . Twój wiek to 24, a Twój dochód wynosi 50000.0 złotych.
```

W przykładowych danych wyjściowych zwróć uwagę na to, że program najpierw pozwala użytkownikowi wprowadzić wiek. Instrukcja z wiersza 20. wczytuje z klawiatury wartość typu `int` i zapisuje ją w zmiennej `age`. Następnie użytkownik wpisuje swój dochód. Instrukcja z wiersza 24. wczytuje z klawiatury wartość typu `double` i zapisuje ją w zmiennej `income`. Następnie program wyświetla użytkownikowi pytanie o nazwisko, jednak wydaje się, że instrukcja z wiersza 28. zostaje pominięta. Program nie wczytuje nazwiska z klawiatury. Wynika to z drobnej różnicy w działaniu metody `nextLine` i innych metod klasy `Scanner`.

Gdy użytkownik wpisuje znaki na klawiaturze, są one zapisywane w obszarze pamięci nazywanym niekiedy *buforem klawiatury*. Wciśnięcie klawisza *Enter* powoduje zapisanie w tym buforze znaku nowego wiersza. W przykładowym programie z listingu 2.30 użytkownik widzi prośbę o wprowadzenie swojego wieku, a instrukcja z wiersza 20. wywołuje metodę `nextInt` w celu wczytania liczby całkowitej z bufora klawiatury. Zauważ, że użytkownik wpisał 24 i wcisnął klawisz *Enter*. Metoda `nextInt` wczytała liczbę całkowitą z bufora klawiatury, a następnie zakończyła działanie po napotkaniu znaku nowego wiersza. Tak więc wartość 24 została wczytana z bufora, ale znak nowego wiersza już nie. Znak ten pozostał w buforze klawiatury.

Następnie program wyświetlił prośbę o wprowadzenie rocznego dochodu użytkownika. Użytkownik wpisał `50000.00`, a następnie wcisnął klawisz *Enter*. Metoda `nextDouble` uruchomiona w wierszu 24. najpierw natrafiła na znak nowego wiersza pozostawiony przez metodę `nextInt`. Nie spowodowało to problemu, ponieważ metoda `nextDouble` jest tak zaprojektowana, aby pomijać napotkane początkowe znaki nowego wiersza. Tak więc metoda ta pominęła początkowy znak nowego wiersza, wczytała wartość `50000.00` z klawiatury i zakończyła wczytywanie danych po napotkaniu następnego znaku nowego wiersza. Ten znak pozostał w buforze klawiatury.

Następnie użytkownik zobaczył prośbę o wprowadzenie nazwiska. W wierszu 28. wywoływana jest metoda `nextLine`. Nie jest ona zaprojektowana pod kątem pomijania początkowego znaku nowego wiersza. Gdy metoda `nextLine` jako pierwszy znak napotyka znak nowego wiersza, nie wczytuje żadnych danych. Ponieważ metoda `nextDouble` z wiersza 24. pozostawiła w buforze klawiatury znak nowego wiersza, metoda `nextLine` nie wczytała żadnych danych wejściowych. Zamiast tego natychmiast zakończyła pracę i użytkownik nie mógł podać nazwiska.

Choć szczegółowy opis tego problemu może wydawać się skomplikowany, jego rozwiązanie jest proste. Na listingu 2.31 pokazano zmodyfikowaną wersję listingu 2.30 z rozwiązanym problemem wprowadzania danych wejściowych.

Listing 2.31 Plik CorrectedInputProblem.java

```
1  import java.util.Scanner; // Potrzebne, by móc używać klasy Scanner.
2
3  /**
4     Ten program poprawnie wczytuje liczbowe i tekstowe dane wejściowe.
5  */
6
7  public class CorrectedInputProblem
8  {
9     public static void main(String[] args)
10    {
11       String name; // Przechowywanie nazwy użytkownika.
12       int age; // Przechowywanie wieku użytkownika.
13       double income; // Przechowywanie dochodu użytkownika.
14
15       // Tworzenie obiektu klasy Scanner w celu wczytywania danych wejściowych.
16       Scanner keyboard = new Scanner(System.in);
17
18       // Pobieranie wieku użytkownika.
19       System.out.print("Ile masz lat? ");
20       age = keyboard.nextInt();
21
22       // Pobieranie dochodu użytkownika.
23       System.out.print("Ile wynosi Twój roczny dochód? ");
24       income = keyboard.nextDouble();
25
26       // Pobieranie pozostałego znaku nowego wiersza.
27       keyboard.nextLine();
28
29       // Pobieranie nazwiska użytkownika.
30       System.out.print("Jak się nazywasz? ");
31       name = keyboard.nextLine();
32
33       // Wyświetlanie informacji użytkownikowi.
34       System.out.println("Witaj, " + name + ". Twój wiek to " +
35                          age + ", a Twój dochód wynosi " +
36                          income + " złotych.");
37    }
38 }
```

Dane wyjściowe programu (przykładowe dane wejściowe są wyróżnione pogrubieniem)

```
Ile masz lat? 24 [Enter]
Ile wynosi Twój roczny dochód? 50000.00 [Enter]
Jak się nazywasz? Marta Suska [Enter]
Witaj, Marta Suska. Twój wiek to 24, a Twój dochód wynosi 50000.0 złotych.
```

Zauważ, że po wczytaniu przez metodę `nextDouble` (wiersz 24.) dochodu użytkownika wywoływana jest metoda `nextLine` (wiersz 27.). Celem tego wywołania jest „zużycie" (usunięcie) znaku nowego wiersza pozostałego w buforze klawiatury. Dalej, w wierszu 31., ponownie wywoływana jest metoda `nextLine`. Tym razem poprawnie wczytuje ona nazwisko użytkownika.

UWAGA: Zauważ, że w wierszu 27., gdzie pobierany jest pozostały znak nowego wiersza, wartość zwracana przez metodę nie jest przypisywana do żadnej zmiennej. Dzieje się tak, ponieważ metoda jest wywoływana tylko w celu usunięcia znaku nowego wiersza, nie trzeba więc zachowywać zwracanej przez nią wartości.

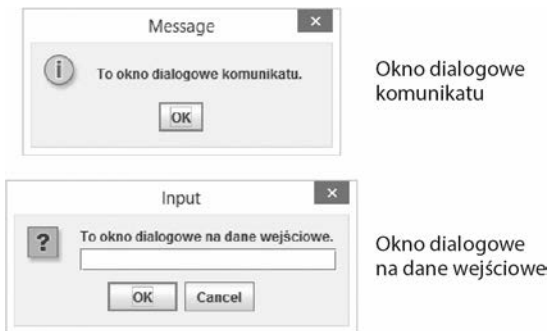
2.14. Okna dialogowe

WYJAŚNIENIE: Klasa `JOptionPane` umożliwia szybkie wyświetlanie okna dialogowego, czyli niewielkiego okna graficznego wyświetlającego komunikat lub żądanie wprowadzenia danych wejściowych.

Okno dialogowe to niewielkie okno graficzne wyświetlające komunikat lub prośbę o dane wejściowe. Do szybkiego wyświetlania takich okien można posłużyć się klasą `JOption` ↪`Pane`. W tym podrozdziale omówimy wymienione niżej typy okien dialogowych i sposoby wyświetlania ich za pomocą klasy `JOptionPane`.

- Okno dialogowe komunikatu — Okno dialogowe wyświetlające komunikat. Obejmuje także przycisk *OK*.
- Okno dialogowe na dane wejściowe — Okno dialogowe wyświetlające prośbę o dane wejściowe. Obejmuje pole tekstowe, gdzie dane są wprowadzane. Zawiera też przyciski *OK* i *Cancel*.

Na rysunku 2.14 pokazano przykładowe okna dialogowe obu rodzajów.



Rysunek 2.14. Okno dialogowe komunikatu i okno dialogowe na dane wejściowe

Klasa `JOptionPane` nie jest automatycznie dostępna w programach Javy. W każdym programie, w którym używana jest ta klasa, w początkowej części pliku trzeba umieścić następującą instrukcję:

```
import javax.swing.JOptionPane;
```

Informuje ona kompilator o tym, gdzie znajduje się klasa `JOptionPane` i zapewnia jej dostępność w programie.

Wyświetlanie okien dialogowych

Do wyświetlania okna dialogowego służy metoda `showMessageDialog`. Oto instrukcja ją wywołująca:

```
JOptionPane.showMessageDialog(null, "Witaj, świecie!");
```

Pierwszy podany argument jest istotny tylko w programach wyświetlających inne okna graficzne. W prezentowanych przykładach pierwszym argumentem zawsze jest słowo kluczowe `null`. To powoduje, że okno dialogowe jest wyświetlane pośrodku ekranu. Drugim argumentem jest komunikat, który ma się pojawić w oknie dialogowym. Pokazany kod powoduje wyświetlenie okna dialogowego z rysunku 2.15. Gdy użytkownik kliknie przycisk *OK*, okno dialogowe zostanie zamknięte.

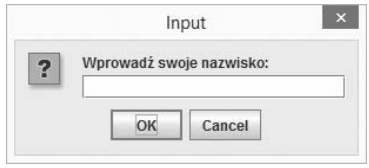
Rysunek 2.15. Okno dialogowe (Oracle Corporate Counsel)

Wyświetlanie okien dialogowych na dane wejściowe

Okno dialogowe na dane wejściowe to szybki i prosty sposób, by poprosić użytkownika o wpisanie danych. Do wyświetlania takich okien służy metoda `showInputDialog` klasy `JOptionPane`. Poniższy kod wywołuje tę metodę:

```
String name;
name = JOptionPane.showInputDialog("Wprowadź swoje nazwisko:");
```

Argument przekazany do tej metody to komunikat, jaki ma się pojawić w oknie dialogowym. Ta instrukcja powoduje wyświetlenie pośrodku ekranu okna dialogowego z rysunku 2.16. Jeśli użytkownik kliknie przycisk *OK*, zmienna `name` będzie wskazywać tekst wprowadzony przez użytkownika w polu tekstowym. Jeżeli użytkownik kliknie przycisk *Cancel*, zmienna `name` będzie wskazywać wartość specjalną `null`.



Rysunek 2.16. Okno dialogowe (Oracle Corporate Counsel)

Przykładowy program

W programie z listingu 2.32 pokazano, jak posługiwać się oknami dialogowymi obu typów. W tym programie okna dialogowe na dane wejściowe służą do wyświetlenia użytkownikowi próśb o podanie imienia i nazwiska. W trakcie działania tego programu wyświetlane są jedno po drugim okna dialogowe z rysunku 2.17.

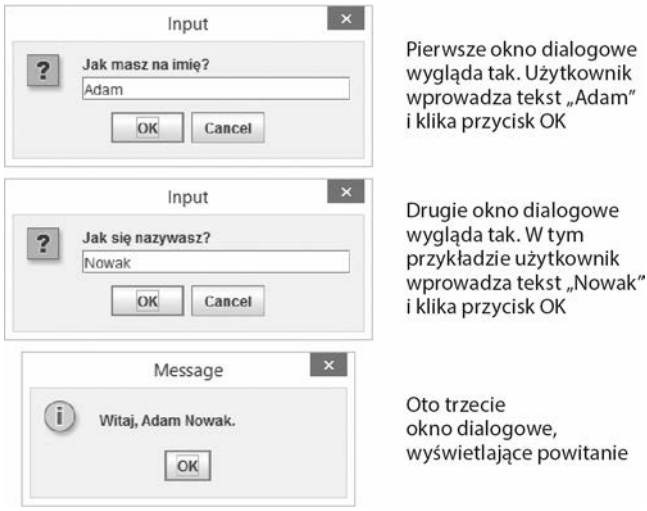
Listing 2.32 Plik NamesDialog.java

```
1  import javax.swing.JOptionPane;
2
3  /**
4     Ten program ilustruje stosowanie okien dialogowych
5     przy użyciu klasy JOptionPane.
6  */
7
8  public class NamesDialog
9  {
10   public static void main(String[] args)
11   {
12     String firstName, lastName;
13
14     // Pobieranie imienia użytkownika.
15     firstName =
16       JOptionPane.showInputDialog("Jak masz na imię?");
17
18     // Pobieranie nazwiska użytkownika.
19     lastName =
20       JOptionPane.showInputDialog("Jak się nazywasz?");
21
22     // Wyświetlanie powitania.
23     JOptionPane.showMessageDialog(null, "Witaj, " + firstName +
24                                   " " + lastName + ".");
25     System.exit(0);
26   }
27 }
```

Zwróć uwagę na ostatnią instrukcję w metodzie `main`:

```
System.exit(0);
```

Ta instrukcja powoduje zakończenie pracy programu. Jest ona niezbędna, gdy klasa `JOptionPane` jest używana w programie konsolowym. Gdy program konsolowy korzysta z klasy `JOptionPane` do wyświetlania okna dialogowego, program nie kończy automatycznie pracy w momencie dotarcia do końca metody `main`. Dzieje się tak, ponieważ



Rysunek 2.17. Okna dialogowe wyświetlane przez program NamesDialog (Oracle Corporate Counsel)

klasa `JOptionPane` powoduje uruchomienie w maszynie JVM dodatkowego wątku. Jeśli metoda `System.exit` nie zostanie wywołana, to zadanie (nazywane także *wątkiem*) będzie kontynuowało pracę nawet po dotarciu do końca metody `main`.

OSTRZEŻENIE! Z metody `System.exit` korzystaj tylko wtedy, gdy jest to bezwzględnie konieczne. W dużym programie metoda `System.exit` może spowodować problemy, ponieważ skutkuje bezwarunkowym zamknięciem maszyny JVM z pominięciem normalnego logicznego przepływu sterowania w programie. W tej książce tę metodę wywołano tylko dlatego, że konieczne jest zamknięcie programu konsolowego używającego klasy `JOptionPane`.

Metoda `System.exit` wymaga argumentu całkowitoliczbowego. Jest nim kod wyjścia przekazywany do systemu operacyjnego. Choć ten kod zwykle jest ignorowany, można go wykorzystać poza programem, aby określić, czy program zakończył pracę z powodzeniem, czy z powodu błędu. Wartość 0 zwyczajowo oznacza udane zakończenie działania.

Przekształcanie tekstowych danych wejściowych na liczby

Klasa `JOptionPane` (w odróżnieniu od klasy `Scanner`) nie udostępnia różnych metod do wczytywania danych wejściowych różnych typów. Metoda `showInputDialog` zawsze zwraca dane wejściowe od użytkownika jako wartość typu `String` — nawet jeśli użytkownik wprowadzi dane liczbowe. Przykładowo, gdy użytkownik wprowadzi w oknie dialogowym liczbę 72, metoda `showInputDialog` zwróci łańcuch znaków `"72"`. To może być problem, jeśli chcesz wykorzystać dane wejściowe od użytkownika w operacji

matematycznej. Wynika to z tego, że nie można wykonywać takich operacji na łańcuchach znaków. W takim scenariuszu trzeba przekształcić dane wejściowe na wartość liczbową. Do przekształcania wartości tekstowych na liczbowe można posłużyć się jedną z metod wymienionych w tabeli 2.18.

Tabela 2.18. Metody do konwersji łańcuchów znaków na liczby

Metoda	Służy do…	Przykładowy kod
`Byte.parseByte`	przekształcania łańcucha znaków na typ `byte`.	`byte num;` `num = Byte.parseByte(str);`
`Double.parseDouble`	przekształcania łańcucha znaków na typ `double`.	`double num;` `num = Double.parseDouble(str);`
`Float.parseFloat`	przekształcania łańcucha znaków na typ `float`.	`float num;` `num = Float.parseFloat(str);`
`Integer.parseInt`	przekształcania łańcucha znaków na typ `int`.	`int num;` `num = Integer.parseInt(str);`
`Long.parseLong`	przekształcania łańcucha znaków na typ `long`.	`long num;` `num = Long.parseLong(str);`
`Short.parseShort`	przekształcania łańcucha znaków na typ `short`.	`short num;` `num = Short.parseShort(str);`

UWAGA: Metody z tabeli 2.18 należą do klas nakładkowych Javy. Więcej na ich temat dowiesz się z rozdziału 9.

Oto przykład pokazujący, jak używać metody `Integer.parseInt` do przekształcania wartości zwracanej przez metodę `JOptionPane.showInputDialog` na typ `int`:

```
int number;
String str;
str = JOptionPane.showInputDialog("Wprowadź liczbę:");
number = Integer.parseInt(str);
```

Po wykonaniu tego kodu w zmiennej `number` znajdzie się wprowadzona przez użytkownika wartość przekształcona na typ `int`. Oto przykład pokazujący, jak używać metody `Double.parseDouble` do przekształcania danych wejściowych od użytkownika na typ `double`:

```
double price;
String str;
str = JOptionPane.showInputDialog("Wprowadź cenę detaliczną:");
price = Double.parseDouble(str);
```

Po wykonaniu tego kodu w zmiennej `price` znajdzie się wartość wprowadzona przez użytkownika, przekształcona na typ `double`. Na listingu 2.33 pokazany jest kompletny program. Jest to zmodyfikowana wersja programu *Payroll.java* z listingu 2.29. Wykonanie tego programu powoduje wyświetlenie jedno po drugim okien dialogowych z rysunku 2.18.

Listing 2.33 Plik PayrollDialog.java

```
1  import javax.swing.JOptionPane;
2
3  /**
4     Ten program ilustruje używanie okien dialogowych
5     za pomocą klasy JOptionPane.
6  */
7
8  public class PayrollDialog
9  {
10   public static void main(String[] args)
11   {
12     String inputString; // Do wczytywania danych wejściowych.
13     String name;        // Nazwisko użytkownika.
14     int hours;          // Liczba przepracowanych godzin.
15     double payRate;     // Stawka godzinowa użytkownika.
16     double grossPay;    // Pensja brutto użytkownika.
17
18     // Pobieranie nazwiska użytkownika.
19     name = JOptionPane.showInputDialog("Jak się " +
20                                        "nazywasz? ");
21
22     // Pobieranie liczby przepracowanych godzin.
23     inputString =
24       JOptionPane.showInputDialog("Ile godzin " +
25                                   "przepracowałeś w tym tygodniu? ");
26
27     // Przekształcanie danych wejściowych na typ int.
28     hours = Integer.parseInt(inputString);
29
30     // Pobieranie stawki godzinowej.
31     inputString =
32       JOptionPane.showInputDialog("Ile wynosi " +
33                                   "Twoja stawka godzinowa? ");
34
35     // Przekształcanie danych wejściowych na typ double.
36     payRate = Double.parseDouble(inputString);
37
38     // Obliczanie pensji brutto.
39     grossPay = hours * payRate;
40
41     // Wyświetlanie wyników.
42     JOptionPane.showMessageDialog(null, "Witaj, " +
43                     name + ". Twoja pensja brutto wynosi " +
44                     grossPay + " złotych.");
45
46     // Koniec programu.
47     System.exit(0);
48   }
49 }
```

Punkt kontrolny

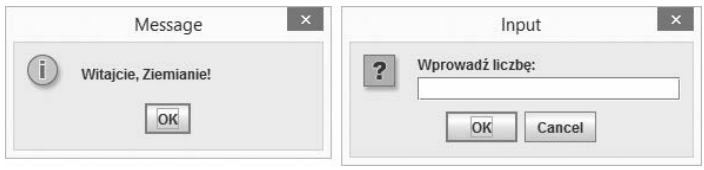
2.34. Jakie jest przeznaczenie następujących rodzajów okien dialogowych:

okna dialogowe komunikatu,

okna dialogowego na dane wejściowe?

Rysunek 2.18. Okna dialogowe wyświetlane przez program PayrollDialog.java (Oracle Corporate Counsel)

2.35. Napisz kod, który wyświetli okna dialogowe widoczne na rysunku 2.19.



Rysunek 2.19. Okna dialogowe (Oracle Corporate Counsel)

2.36. Napisz kod wyświetlający okno dialogowe z pytaniem o wiek użytkownika. Kod powinien przekształcać wprowadzoną wartość na typ `int` i zapisywać ją w zmiennej `age` typu `int`.

2.37. Jaką instrukcję `import` zastosujesz w programie używającym klasy `JOptionPane`?

2.15. Typowe błędy, których należy unikać

- **Niedopasowane nawiasy klamrowe, cudzysłowy, apostrofy lub zwykłe nawiasy.** W tym rozdziale zobaczyłeś, że instrukcje tworzące definicję klasy znajdują się w nawiasach klamrowych. Ponadto zobaczyłeś, że instrukcje w metodzie także

są umieszczane w takich nawiasach. Każdemu nawiasowi otwierającemu musi odpowiadać nawias zamykający we właściwym miejscu. To samo dotyczy cudzysłowów obejmujących literały tekstowe i apostrofów zawierających znaki. Ponadto w instrukcjach z nawiasami, np. w wyrażeniach matematycznych, trzeba dodać nawias zamykający dla każdego nawiasu otwierającego.

- **Błędnie zapisane słowa kluczowe.** Java nie rozpoznaje błędnie zapisanych słów kluczowych.
- **Używanie wielkich liter w słowach kluczowych.** Pamiętaj, że w Javie wielkość liter ma znaczenie, a wszystkie słowa kluczowe są pisane małymi literami. Wielkie litery w słowach kluczowych to błędny zapis.
- **Używanie słów kluczowych w nazwach zmiennych.** Słowa kluczowe są zarezerwowane do specjalnego użytku. Nie można ich stosować w żadnym innym celu.
- **Niespójna pisownia nazw zmiennych.** Za każdym razem, gdy używasz nazwy zmiennej, musi być ona zapisana dokładnie w taki sposób jak w deklaracji tej zmiennej.
- **Niespójna wielkość liter w nazwach zmiennych.** Ponieważ w Javie wielkość liter ma znaczenie, wielkie litery są traktowane jako różne od małych. Java nie rozpozna nazwy, która nie jest zapisana dokładnie w taki sam sposób jak w deklaracji zmiennej.
- **Wstawianie spacji w nazwach zmiennych.** Spacje w nazwach zmiennych są niedozwolone. Zamiast stosować dwuwyrazowe nazwy, np. `gross pay`, należy posługiwać się pojedynczymi słowami, np. `grossPay`.
- **Pominięcie średnika na końcu instrukcji.** Każda kompletna instrukcja w Javie kończy się średnikiem.
- **Przypisanie literału typu `double` do zmiennej typu `float`.** Java jest językiem ze ścisłą kontrolą typów. To oznacza, że w zmiennych można zapisywać tylko wartości zgodnych typów danych. Wszystkie literały zmiennoprzecinkowe są traktowane jako typu `double`, a wartość tego typu nie jest zgodna ze zmienną typu `float`. Literały zmiennoprzecinkowe muszą kończyć się literą `f` lub `F`, aby można je było zapisać w zmiennych typu `float`.
- **Stosowanie przecinków lub symboli walut w literałach liczbowych.** Literały liczbowe nie mogą zawierać przecinków ani symboli walut (np. znaku dolara).
- **Przypadkowe wykonanie dzielenia całkowitoliczbowego.** Gdy oba operandy instrukcji dzielenia są liczbami całkowitymi, instrukcja zwróci liczbę całkowitą. Ewentualna reszta jest odrzucana.
- **Niepogrupowanie części wyrażenia matematycznego.** Jeśli w wyrażeniu matematycznym występuje więcej niż jeden operator, to wyrażenie jest przetwarzane zgodnie z kolejnością operacji. Jeżeli chcesz zmienić kolejność działania operatorów, musisz zastosować nawiasy do pogrupowania części wyrażenia.
- **Wstawianie spacji w złożonym operatorze przypisania.** Między dwoma operatorami tworzącymi złożony operator przypisania nie może występować spacja.

- **Używanie zmiennej do zapisywania wyniku obliczeń, gdy typ danych tej zmiennej jest niezgodny z typem danych wyniku.** Zmienna, w której zapisywany jest wynik obliczeń, musi mieć typ danych zgodny z typem danych wyniku.
- **Błędne zakończenie komentarza wielowierszowego lub komentarza `javadoc`.** Komentarze wielowierszowe i komentarze `javadoc` kończą się znakami `*/`. Jeśli zapomnisz umieścić te znaki w odpowiednim punkcie zakończenia komentarza lub przypadkowo zamienisz znaki `*` i `/` miejscami, komentarz nie będzie miał punktu zakończenia.
- **Niezastosowanie właściwej instrukcji `import` w programie korzystającym z klas `Scanner` lub `JOptionPane`.** Aby klasa `Scanner` była dostępna w programie, trzeba zastosować instrukcję `import java.util.Scanner;` w początkowej części tego programu. Aby w programie dostępna była klasa `JOptionPane`, w jego początkowej części należy umieścić instrukcję `import javax.swing.JOptionPane;`.
- **Nieprzekształcenie wartości zwróconej przez metodę `showInputDialog` na liczbę, gdy używa się okna dialogowego na dane wejściowe do wczytywania danych liczbowych.** Metoda `showInputDialog` zawsze zwraca dane wejściowe od użytkownika jako łańcuch znaków. Jeśli użytkownik wprowadzi wartość liczbową, trzeba ją przekształcić na liczbę; dopiero potem można z niej korzystać w instrukcjach matematycznych.

Pytania kontrolne i ćwiczenia

Pytania wielokrotnego wyboru i pytania typu prawda/fałsz

1. Każda kompletna instrukcja kończy się:
   a. kropką,
   b. nawiasami,
   c. średnikiem,
   d. końcowym nawiasem klamrowym.
2. Poniższe dane:

   ```
   72
   'A'
   "Witaj, świecie!"
   2.8712
   ```

   to przykładowe:
   a. zmienne,
   b. literały,
   c. łańcuchy znaków,
   d. żadne z powyższych.
3. Grupa instrukcji, np. zawartość klasy lub metody, jest umieszczona w:
   a. nawisach klamrowych {},
   b. nawiasach zwykłych (),
   c. nawiasach kwadratowych [],
   d. nawiasach dowolnego rodzaju.

4. Które z poniższych wyrażeń *nie* jest poprawną instrukcją przypisania? Zaznacz wszystkie nieprawidłowe instrukcje.
   a. `total = 9;`,
   b. `72 = amount;`,
   c. `profit = 129`,
   d. `letter = 'W';`.
5. Które z poniższych wyrażeń *nie* jest poprawną instrukcją `println`? Zaznacz wszystkie nieprawidłowe instrukcje.
   a. `System.out.println + "Witaj, świecie!";`,
   b. `System.out.println("Miłego dnia!");`,
   c. `out.System.println(value);`,
   d. `println.out(Programowanie to świetna zabawa!);`.
6. Operator negacji jest:
   a. jednoargumentowy,
   b. dwuargumentowy,
   c. trójargumentowy,
   d. żadne z powyższych.
7. Do deklarowania nazwanych stałych służy następujące słowo kluczowe:
   a. `constant`,
   b. `namedConstant`,
   c. `final`,
   d. `concrete`.
8. Początek komentarza wielowierszowego tworzą znaki:
   a. //,
   b. /*,
   c. */,
   d. /**.
9. Początek komentarza jednowierszowego tworzą znaki:
   a. //,
   b. /*,
   c. */,
   d. /**.
10. Początek komentarzy `javadoc` tworzą znaki:
    a. //,
    b. /*,
    c. */,
    d. /**.
11. Którą z metod klasy `Scanner` zastosujesz do wczytywania danych wejściowych w postaci łańcucha znaków?
    a. `nextString`,
    b. `nextLine`,
    c. `readString`,
    d. `getLine`.

12. Którą z metod klasy `Scanner` zastosujesz do wczytywania danych wejściowych typu `double`?
    a. `nextDouble`,
    b. `getDouble`,
    c. `readDouble`,
    d. żadne z powyższych — za pomocą klasy `Scanner` nie można wczytywać wartości typu `double`.
13. Do wyświetlania okien dialogowych można zastosować klasę:
    a. `JOptionPane`,
    b. `BufferedReader`,
    c. `InputStreamReader`,
    d. `DialogBox`.
14. Gdy Java przekształca wartość typu z niższego poziomu hierarchii na typ z wyższego poziomu hierarchii, operacja ta jest nazywana:
    a. konwersją 4-bitową,
    b. konwersją eskalacyjną,
    c. konwersją poszerzającą zakres,
    d. konwersją zawężającą zakres.
15. Ten rodzaj operatora umożliwia ręczne przekształcanie wartości, nawet jeśli skutkuje to konwersją zawężającą zakres:
    a. operator rzutowania,
    b. operator dwuargumentowy,
    c. operator wczytywania,
    d. operator kropki.
16. **Prawda czy fałsz?** W programach w Javie lewemu nawiasowi klamrowemu zawsze odpowiada prawy nawias klamrowy w dalszej części kodu.
17. **Prawda czy fałsz?** Zmienną trzeba zadeklarować przed jej użyciem.
18. **Prawda czy fałsz?** Nazwy zmiennych mogą rozpoczynać się cyfrą.
19. **Prawda czy fałsz?** Nie można zmienić wartości zmiennej, jeśli w jej deklaracji zastosowano słowo kluczowe `final`.
20. **Prawda czy fałsz?** Komentarze rozpoczynające się znakami // mogą być przetwarzane przez program `javadoc`.
21. **Prawda czy fałsz?** Jeśli jeden z operandów operatora jest typu `double`, a drugi typu `int`, Java automatycznie przekształci wartość typu `double` na typ `int`.

Przewidywanie danych wyjściowych

Co poniższe fragmenty kodu wyświetlą na ekranie?

1.
```
int freeze = 32, boil = 212;
freeze = 0;
boil = 100;
System.out.println(freeze + "\n"+ boil + "\n");
```

2.
```
int x = 0, y = 2;
x = y * 4;
System.out.println(x + "\n"+ y + "\n");
```

3.
```
System.out.print("Jestem niezwykłą");
System.out.print("maszyną\nobliczeniową");
System.out.print("\ni\nzaskoczę\n)");
System.out.println("Cię.");
```

4.
```
System.out.print("Uważaj\n)");
System.out.print("To pytanie/n może być ");
System.out.println("podchwytliwe.");
```

5.
```
int a, x = 23;
a = x % 2;
System.out.println(x + "\n"+ a);
```

Znajdź błąd

W poniższym programie występuje wiele błędów składni. Znajdź ich tyle, ile potrafisz.

```
*/ Jakie błędy występują w tym programie? /*
public MyProgram
{
  public static void main(String[] args);
  }
    int a, b, c \\ Trzy liczby całkowite
    a = 3
    b = 4
    c = a + b
    System.out.println('Wartość zmiennej c to' + C);
  {
```

Warsztat projektanta algorytmów

1. Pokaż, jak za pomocą jednej instrukcji zadeklarować zmienne `temp`, `weight` i `age` typu `double`.
2. Pokaż, jak w jednej instrukcji zadeklarować zmienne `months`, `days` i `years` typu `int`, przy czym zmienną `months` zainicjuj wartością 2, a zmienną `years` wartością 3.
3. Napisz instrukcje przypisania, które wykonują następujące operacje na zmiennych `a`, `b` i `c`:
   a. dodawanie 2 do `a` i zapisywanie wyniku w `b`,
   b. mnożenie `b` przez 4 i zapisywanie wyniku w `a`,
   c. dzielenie `a` przez 3,14 i zapisywanie wyniku w `b`,
   d. odejmowanie 8 od `b` i zapisywanie wyniku w `a`,
   e. zapisywanie litery `'K'` w `c`,
   f. zapisywanie kodu Unicode litery `'B'` w `c`.
4. Załóżmy, że zmienne `result`, `w`, `x`, `y` i `z` to liczby całkowite, przy czym `w = 5`, `x = 4`, `y = 8` i `z = 2`. Jaka wartość zostanie zapisana w zmiennej `result` w wyniku wykonania poniższych instrukcji?
   a. `result = x + y;`,
   b. `result = z * 2;`,

c. `result = y / x;`,
d. `result = y - z;`,
e. `result = w % 2;`.

5. Jak każda z poniższych liczb jest reprezentowana w notacji E?
   a. $3{,}287 \times 10^{6}$,
   b. $-9{,}7865 \times 10^{12}$,
   c. $7{,}65491 \times 10^{-3}$.

6. Zmodyfikuj poniższy program, aby wyświetlał dwa puste wiersze między poszczególnymi wierszami tekstu.

```
public class
{
  public static void main(String[] args)
  {
    System.out.print("Umrzeć – tego nie robi się kotu.");
    System.out.print("Bo co ma począć kot");
    System.out.print("w pustym mieszkaniu.");
    System.out.println("                    – Wisława Szymborska");
  }
}
```

7. Co wyświetli poniższy kod?

```
int apples = 0, bananas = 2, pears = 10;
apples += 10;
bananas *= 10;
pears /= 10;
System.out.println(apples + " " +
                   bananas + " " +
                   pears);
```

8. Co wyświetli poniższy kod?

```
double d = 12.9;
int i = (int)d;
System.out.println(i);
```

9. Co wyświetli poniższy kod?

```
String message = "Udanego dnia!";
System.out.println(message.charAt(5));
```

10. Co wyświetli poniższy kod?

```
String message = "Udanego dnia!";
System.out.println(message.toUpperCase());
System.out.println(message);
```

11. Przekształć przedstawiony poniżej pseudokod na kod w Javie. Pamiętaj, aby zadeklarować odpowiednie zmienne.

Zapisz 20 w zmiennej speed.
Zapisz 10 w zmiennej time.
Pomnóż przez siebie zmienne speed i time, a wynik zapisz w zmiennej distance.
Wyświetl zawartość zmiennej distance.

12. Przekształć poniższy pseudokod na kod w Javie. Pamiętaj, aby zadeklarować odpowiednie zmienne.

Zapisz 172,5 w zmiennej force.
Zapisz 27,5 w zmiennej area.
Podziel zmienną area przez zmienną force, a wynik zapisz w zmiennej pressure.
Wyświetl zawartość zmiennej pressure.

13. Napisz kod przygotowujący wszystkie obiekty potrzebne do odczytu danych z klawiatury. Następnie napisz kod, który wyświetla prośbę o wpisanie przez użytkownika oczekiwanego rocznego dochodu. Wprowadzone dane należy zapisać w zmiennej typu `double`.
14. Napisz kod wyświetlający okno dialogowe z prośbą o wprowadzenie przez użytkownika oczekiwanego rocznego dochodu. Dane wejściowe należy zapisać w zmiennej typu `double`.
15. Program zawiera zmienną `total` typu `float` i zmienną `number` typu `double`. Napisz instrukcję, która przypisuje wartość zmiennej `number` do zmiennej `total`, nie powodując błędu w trakcie kompilacji.

Krótkie odpowiedzi

1. Czy poniższy komentarz ma składnię jednowierszową, czy wielowierszową?
   ```
   /* Ten program został napisany przez M. A. Kodera. */
   ```
2. Czy poniższy komentarz ma składnię jednowierszową, czy wielowierszową?
   ```
   // Ten program został napisany przez M. A. Kodera.
   ```
3. Opisz, co oznacza pojęcie „program samodokumentujący się”.
4. Co oznacza, że w języku „istotna jest wielkość liter”? Dlaczego dla programisty ważne jest, by wiedział, czy w Javie istotna jest wielkość liter?
5. Pokrótce wyjaśnij, jak metody `print` i `println` są powiązane z klasą `System` i obiektem `out`.
6. O jakich cechach zmiennej jej deklaracja informuje kompilator Javy?
7. Dlaczego używanie dla zmiennych nazw takich jak `x` nie jest zalecane?
8. Co trzeba uwzględnić w momencie wyboru typu danych zmiennej?
9. Pokrótce opisz różnice między przypisywaniem wartości do zmiennej a inicjowaniem zmiennej.
10. Czym różnią się komentarze rozpoczynające się od sekwencji // od komentarzy zaczynających się znakami /*?
11. Opisz pokrótce, czym jest styl programowania. Dlaczego należy zachować jego spójność?
12. Załóżmy, że w programie używana jest nazwana stała `PI` reprezentująca wartość `3.14`. Jest ona wykorzystywana w kilku miejscach programu. Jakie zalety daje używanie tej stałej zamiast wartości `3.14` w każdej instrukcji?
13. Przyjmijmy, że *SalesAverage.java* to plik źródłowy w Javie zawierający komentarze `javadoc`. Załóżmy ponadto, że przeszedłeś do katalogu zawierającego ten plik. Jaką instrukcję wpiszesz w wierszu poleceń systemu operacyjnego, aby wygenerować pliki HTML z dokumentacją?

14. Wyrażenie dodaje do siebie zmienne typów `byte` i `short`. Jaki będzie typ danych wyniku?

Zadania programistyczne

1. Nazwisko, wiek i roczny dochód

Napisz program w którym zadeklarowane są:

- zmienna `name` typu `String`,
- zmienna `age` typu `int`,
- zmienna `annualPay` typu `double`.

Zapisz w tych zmiennych literały reprezentujące Twoje nazwisko, wiek i oczekiwany roczny dochód. Program powinien wyświetlać te wartości na ekranie w sposób podobny do poniższego:

```
Nazywam się Adam Kowalski, mam 26 lat
i chcę zarabiać 100000.0 złotych rocznie.
```

2. Nazwisko i inicjały

Napisz program zawierający następujące zmienne typu `String`: `firstName`, `middleName` i `lastName`. Zainicjuj te zmienne swoim pierwszym imieniem, drugim imieniem i nazwiskiem. Program powinien ponadto obejmować następujące zmienne typu `char`: `first` ↪`Initial`, `middleInitial` i `lastInitial`. Zapisz w tych zmiennych inicjały pierwszego imienia, drugiego imienia i nazwiska. Program powinien wyświetlać zawartość tych zmiennych na ekranie.

3. Dane osobowe

Napisz program, który wyświetla w odrębnych wierszach następujące informacje:

- imię i nazwisko,
- adres (z miastem i kodem pocztowym),
- numer telefonu,
- kierunek studiów.

Choć te elementy powinny być wyświetlone w odrębnych wierszach, w programie zastosuj tylko jedną instrukcję `println`.

4. Rysunek z gwiazdek

Napisz program, który wyświetla następujący rysunek:

```
   *
  ***
 *****
```

```
*******
 *****
  ***
   *
```

5. Prognozy sprzedaży

Region mazowiecki generuje 62% sprzedaży w firmie. Napisz program, który na podstawie tego procentu oszacuje, jaką sprzedaż wygeneruje region mazowiecki, jeśli firma w ciągu roku sprzeda produkty w sumie za 4,6 mln złotych. *Wskazówka: Do reprezentowania 62% zastosuj wartość 0,62.*

6. Obliczenia powierzchni

Jeden akr ziemi to 43 560 stóp kwadratowych. Napisz program, który określa liczbę akrów na obszarze zajmującym 389 767 stóp kwadratowych. *Wskazówka: Podziel powierzchnię w stopach kwadratowych przez wielkość akrów, aby otrzymać liczbę akrów.*

7. Amerykański podatek od sprzedaży

Napisz program, który wyświetla prośbę o wprowadzenie wartości kupowanego produktu. Program ten powinien następnie obliczać podatek stanowy i lokalny od sprzedaży. Załóżmy, że podatek stanowy wynosi 4%, a podatek lokalny to 2%. Program powinien wyświetlać wartość produktu, podatek stanowy, podatek lokalny i łączną cenę sprzedaży (czyli sumę wartości produktu i obu podatków od sprzedaży). *Wskazówka: Posłuż się wartością 0,02, aby przedstawić 2%, i wartością 0,04 do reprezentowania 4%.*

8. Kalorie w ciasteczkach

Pudełko zawiera 40 ciasteczek. Z informacji na opakowaniu wynika, że pudełko obejmuje 10 porcji, a każda porcja ma 300 kalorii. Napisz program, który pozwala użytkownikowi wpisać liczbę zjedzonych ciasteczek, a następnie informuje o liczbie skonsumowanych kalorii.

9. Kilometry na litrze

Liczbę kilometrów przejechanych na litrze paliwa można obliczyć według następującego wzoru:

kilometry na litrze = przejechane kilometry/litry paliwa

Napisz program, który wyświetla prośbę o wprowadzenie liczby przejechanych kilometrów i zużytych litrów paliwa. Program powinien obliczać liczbę kilometrów przejechanych na litrze i wyświetlać wynik na ekranie.

10. Średnia z testów

Napisz program, który wyświetla prośbę o wprowadzenie wyników z trzech testów. Program powinien wyświetlać wyniki z każdego testu, a także średnią.

11. Zyski ze sprzedaży płytek drukowanych

Producent elektroniki sprzedaje płytki drukowane z marżą 40%. Jeśli znasz cenę detaliczną płytki, możesz obliczyć zysk za pomocą następującego wzoru:

zysk = cena detaliczna × 0,4

Napisz program, który prosi o wprowadzenie ceny detalicznej płytki, oblicza zysk ze sprzedaży tego produktu i wyświetla wynik na ekranie.

12. Operacje na łańcuchach znaków

Napisz program, który wyświetla użytkownikowi prośbę o wprowadzenie nazwy ulubionego miasta. Dane wejściowe zapisz w zmiennej typu `String`. Program powinien wyświetlać następujące informacje:

- liczbę znaków w nazwie miasta,
- nazwę miasta zapisaną wielkimi literami,
- nazwę miasta zapisaną małymi literami,
- pierwszy znak nazwy miasta.

13. Rachunek w restauracji

Napisz program, który oblicza podatek i napiwek na podstawie rachunku w restauracji. Program powinien wyświetlać użytkownikowi prośbę o wprowadzenie ceny posiłku. Przyjmij podatek na poziomie 6,75% ceny, a napiwek na poziomie 20% sumy ceny i podatku. Wyświetl na ekranie cenę posiłku, wartość podatku, wysokość napiwku i łączną kwotę do zapłaty.

14. Odsetek kobiet i mężczyzn

Napisz program wyświetlający prośbę o wprowadzenie liczby kobiet i mężczyzn zarejestrowanych na kurs. Program powinien wyświetlać odsetek obu płci na kursie.

Wskazówka: Załóżmy, że w kursie uczestniczy 12 kobiet i 8 mężczyzn, co daje 20 studentów. Odsetek mężczyzn można obliczyć w następujący sposób: 8/20 = 0,4, czyli 40%. Odsetek kobiet wynosi 12/20 = 0,6, czyli 60%.

15. Prowizje od handlu akcjami

Katarzyna kupiła 600 akcji w cenie 21,77 złotego za akcję. Musi zapłacić swojemu brokerowi 2% prowizji za transakcję. Napisz program, który oblicza i wyświetla następujące informacje:

- kwotę zapłaconą za same akcje (bez prowizji),
- wysokość prowizji,
- łączną zapłaconą kwotę (cena akcji plus prowizja).

16. Spożycie napojów energetycznych

Producent napojów bezalkoholowych przeprowadził ankietę z udziałem 12 467 konsumentów i odkrył, że około 14% ankietowanych kupuje przynajmniej jeden napój energetyczny tygodniowo. Około 64% kupujących napoje energetyczne preferuje smak cytrusowy. Napisz program wyświetlający następujące informacje:

- przybliżoną liczbę ankietowanych osób, które kupują przynajmniej jeden napój energetyczny tygodniowo;
- przybliżoną liczbę ankietowanych osób, które preferują napoje energetyczne o smaku cytrusowym.

17. Dostosowywanie ilości składników w przepisie

W przepisie na ciasteczka podana jest następująca ilość składników:

- 1,5 szklanki cukru,
- 1 szklanka masła,
- 2,75 szklanki mąki.

Te składniki pozwalają upiec 48 ciasteczek. Napisz program, który wyświetla użytkownikowi prośbę o wpisanie oczekiwanej liczby ciasteczek, a następnie informuje, ile szklanek poszczególnych składników potrzeba do uzyskania tej liczby.

18. Zabawa słowna

Napisz program będący zabawą słowną. Program ten powinien wyświetlać użytkownikowi prośbę o wprowadzenie następujących informacji:

- swojego imienia,
- swojego wieku,
- nazwy zamieszkiwanej miejscowości,
- nazwy uczelni,
- zawodu,
- gatunku zwierzęcia,
- imienia zwierzęcia.

Po wprowadzeniu przez użytkownika tych informacji program powinien wyświetlać następującą historyjkę, uzupełnioną w odpowiednich miejscach danymi wejściowymi:

```
Pewnego razu żył sobie IMIĘ, który mieszkał w MIASTO. W wieku WIEK IMIĘ rozpoczął
studia na UCZELNIA. IMIĘ ukończył studia i rozpoczął pracę jako ZAWÓD. Wtedy IMIĘ
adoptował GATUNEK o imieniu IMIĘ_ZWIERZĘCIA i żyli razem długo i szczęśliwie!
```

19. Program do obsługi transakcji giełdowych

W zeszłym miesiącu Jacek kupił trochę akcji firmy Acme Software. Oto szczegóły tej transakcji:

- liczba zakupionych akcji: 1000,
- cena zakupu akcji: 32,87 złotego za akcję,
- prowizja wypłacona brokerowi: 2% ceny akcji.

Po dwóch tygodniach Jacek sprzedał akcje. Oto szczegóły transakcji sprzedaży:

- liczba sprzedanych akcji: 1000,
- cena sprzedaży akcji: 33,92 złotego za akcję,
- prowizja wypłacona brokerowi: 2% ceny akcji.

Napisz program wyświetlający następujące informacje:

- kwotę zapłaconą przez Jacka za akcje,
- wartość prowizji zapłaconej przez Jacka brokerowi przy zakupie,
- kwotę otrzymaną przez Jacka ze sprzedaży akcji,
- wartość prowizji zapłaconej przez Jacka brokerowi przy sprzedaży,
- zysk Jacka ze sprzedaży akcji po zapłaceniu obu prowizji (jeśli wysokość zysku wyświetlana przez program jest ujemna, oznacza to, że Jacek poniósł stratę).

20. Sadzenie winogron

Właściciel winnicy sadzi kilka nowych rzędów winogron i musi wiedzieć, ile sadzonek ma umieścić w każdym rzędzie. Po pomiarze rzędów stwierdził, że może zastosować do tego pokazany niżej wzór (uwzględnia on okratowanie, jakie trzeba zbudować po obu końcach każdego rzędu):

$$V = \frac{R - 2E}{S}$$

Oto wartości wykorzystywane w tym wzorze:

- V to liczba sadzonek, jakie można umieścić w rzędzie,
- R to długość rzędu w metrach,
- E to ilość miejsca zajmowanego przez okratowanie,
- S to odległość między sadzonkami w metrach.

Napisz program, który wykonuje potrzebne obliczenia za właściciela winnicy. Program powinien wyświetlać użytkownikowi prośbę o wprowadzenie następujących danych:

- długości rzędu w metrach,
- ilości miejsca zajmowanego przez okratowanie w metrach,
- odległości między sadzonkami w metrach.

Po wprowadzeniu danych wejściowych program powinien obliczać i wyświetlać liczbę sadzonek, jakie zmieszczą się w rzędzie.

21. Procent składany

Gdy bank wypłaca procent składany, płaci odsetki nie tylko od zdeponowanej kwoty, ale też od zakumulowanej z czasem kwoty odsetek. Załóżmy, że chcesz zdeponować pewną kwotę na rachunku oszczędnościowym i pozwolić na akumulowanie procentu składanego przez określoną liczbę lat. Oto wzór na obliczenie stanu konta po określonej liczbie lat:

$$A = P(1+\frac{r}{n})^{nt}$$

Oto wartości z tego wzoru:

- A to kwota pieniędzy na koncie po określonej liczbie lat,
- P to kwota pierwotnie zdeponowana na koncie,
- r to roczna stopa oprocentowania,
- n określa, ile razy w roku odsetki są doliczane do kapitału,
- t to liczba lat.

Napisz program, który przeprowadza takie obliczenia. Program powinien wyświetlać prośbę o wprowadzenie następujących danych:

- kwoty pieniędzy początkowo zdeponowanej na koncie,
- rocznej stopy oprocentowania,
- ile razy w roku odsetki są kapitalizowane (np. przy kapitalizacji miesięcznej wpisz 12, a przy kapitalizacji kwartalnej — 4),
- liczby lat, przez jakie środki będą znajdować się na koncie i generować odsetki.

Po wprowadzeniu danych wejściowych program powinien obliczać i wyświetlać kwotę, jaka znajdzie się na koncie po określonej liczbie lat.

UWAGA: Użytkownik powinien wprowadzić stopę oprocentowania jako liczbę procentów. Przykładowo, stopę 2% należy zapisać jako 2, a nie 0,02. Program powinien więc dzielić dane wejściowe przez 100, aby przenieść przecinek w odpowiednie miejsce.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

P

R

S

Ś

T

U

W

Z

Notatki

>>> JAVA
ZDOBĄDŹ SOLIDNE PODSTAWY I PROGRAMUJ JAK MISTRZ!

Java jest rozbudowanym, elastycznym i wszechstronnym, a przy tym dojrzałym językiem programowania. Pozwala na tworzenie kodu dla prawie wszystkich rodzajów komputerów, również tych sterujących pracą najbardziej wyspecjalizowanych urządzeń. Można go używać do pisania dużych systemów, małych programów, aplikacji mobilnych i aplikacji WWW. Jest znakomitym wyborem dla osób, które postanowiły związać swoją przyszłość zawodową z którąś z wielu gałęzi informatyki. Aby optymalnie wykorzystać potencjał drzemiący w Javie, koniecznie trzeba zadbać o solidne podstawy — zarówno w teorii, jak i w praktyce kodowania.

Oto zaktualizowane i uzupełnione wydanie cenionego podręcznika dla studentów. Książka została pomyślana w taki sposób, aby maksymalnie ułatwić naukę Javy krok po kroku i pozwolić na możliwie szybkie rozpoczęcie samodzielnego kodowania. Znalazło się tu wprowadzenie do wiedzy o sprzęcie, oprogramowaniu, wykonywaniu programów i kompilacji kodu. Dzięki temu nawet zupełnie początkujące osoby zaczną płynnie posługiwać się typami danych, zmiennymi czy instrukcjami sterującymi. Bardzo starannie omówiono takie tematy jak klasy, obiekty i dziedziczenie. Nie zabrakło wprowadzenia do pracy z bazami danych, plikami czy też podstaw budowy graficznego interfejsu użytkownika za pomocą nowej biblioteki JavaFX.

W tej książce między innymi:

- solidne podstawy programowania i wprowadzenie do Javy
- operacje wejścia-wyjścia, przetwarzanie tekstu oraz obsługa wyjątków
- rekurencja i jej zastosowanie w rozwiązywaniu problemów
- praca z animacjami, dźwiękiem i wideo
- wykorzystanie interfejsu JDBC do pisania aplikacji bazodanowych

Tony Gaddis

jest głównym autorem świetnie przyjętej serii *Starting Out with (Owoce programowania*, Helion), a ponadto, od przeszło dwóch dekad, wykładowcą akademickim, wielokrotnie nagradzanym — otrzymał tytuł Nauczyciela Roku w North Carolina Community Colleges oraz nagrodę Teaching Excellence przyznawaną przez National Institute for Staff and Organizational Development. Autor i współautor książek dotyczących nauki języków: C++, C#, Java, Visual Basic, Python.

ISBN 978-83-283-4829-5

Cena: 149,00 zł

PEARSON
ALWAYS LEARNING