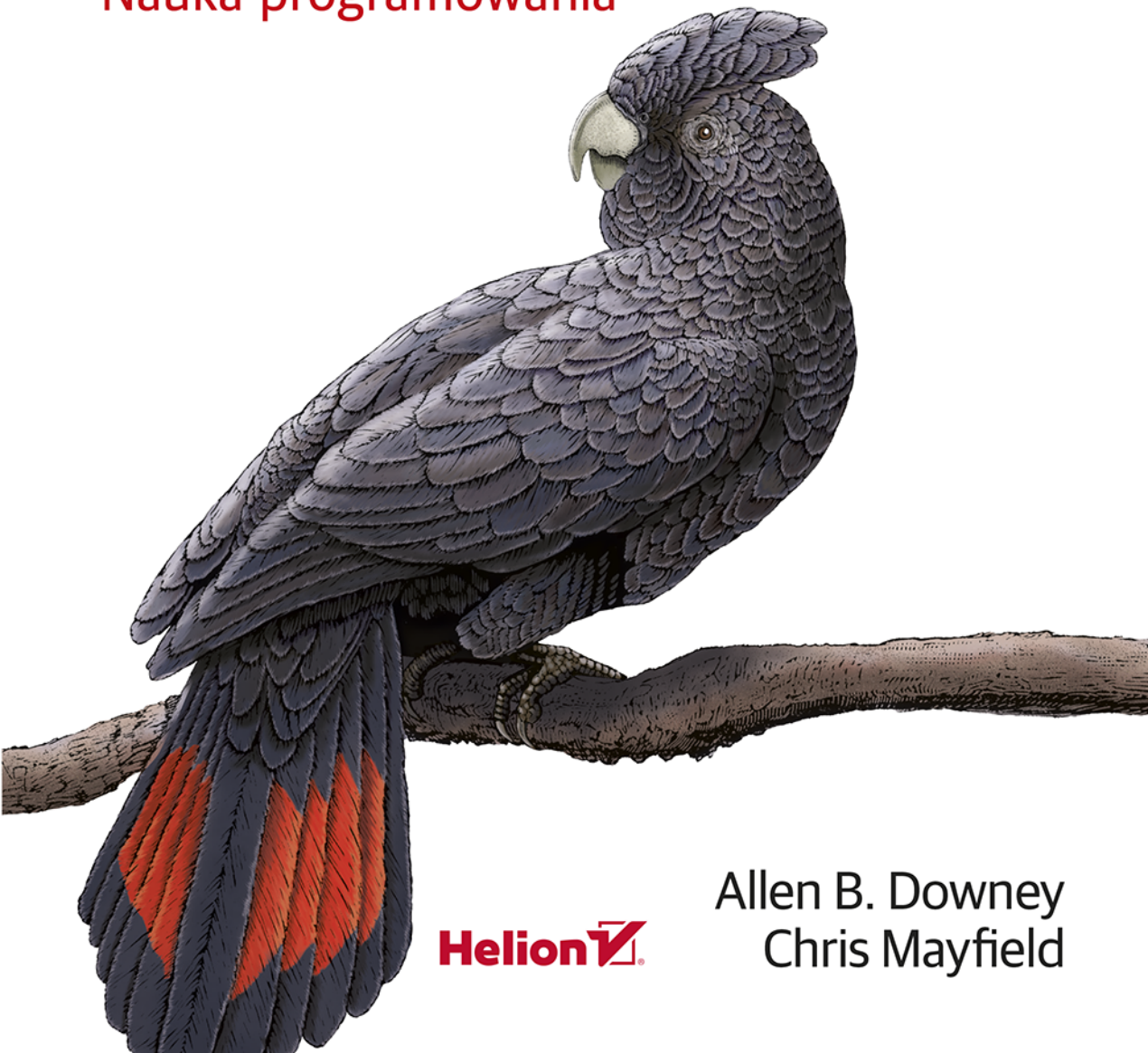
O'REILLY®

Wydanie II

Myśl w języku Java!

Nauka programowania



Helion

Allen B. Downey
Chris Mayfield

Tytuł oryginału: Think Java: How to Think Like a Computer Scientist, 2nd Edition

Tłumaczenie: Łukasz Suma

ISBN: 978-83-283-6719-7

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/mysja2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/mysja2.zip

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Programowanie komputerowe

Celem tej książki jest nauczenie Cię takiego sposobu myślenia, jakim w swojej pracy posługują się informatycy. Sposób ten łączy w sobie niektóre z najlepszych aspektów matematyki, inżynierii i nauk przyrodniczych. Informatycy, podobnie jak matematycy, używają języków formalnych do zapisu swoich pomysłów, a szczególnie do zapisu obliczeń. Podobnie jak inżynierowie, projektują oni pewne rzeczy, tak składając komponenty, aby tworzyły systemy, i dokonując oceny kompromisów wiążących się z określonymi alternatywami. I podobnie jak naukowcy, obserwują działanie skomplikowanych systemów, formułują hipotezy i sprawdzają swoje przypuszczenia.

Ważną umiejętnością w zawodzie informatyka jest umiejętność **rozwiązywania problemów**. Wymaga ona zdolności ich definiowania, kreatywnego myślenia o rozwiązaniach oraz jasnego i precyzyjnego formułowania tych rozwiązań. Jak się okazuje, proces nauki programowania komputerów stanowi doskonałą okazję do rozwoju umiejętności rozwiązywania problemów. Czytając tę książkę, na jednym poziomie będziesz się uczył sztuki pisania programów w języku Java, a więc umiejętności, która sama w sobie jest bardzo przydatna. Jednak na innym będziesz korzystał z programowania jako środka do osiągnięcia pewnego celu. Wraz z postępem lektury cel ten będzie się dla Ciebie stawał coraz bardziej wyraźny.

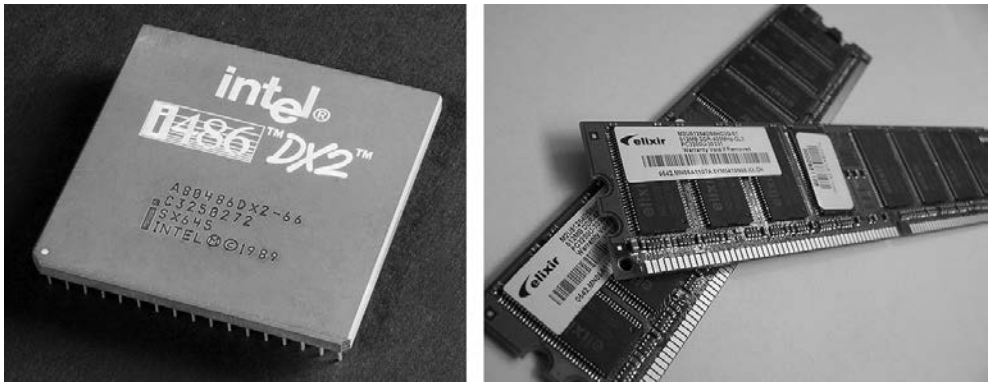
Czym jest komputer?

Gdy ludzie słyszą słowo „komputer", zwykle myślą o laptopie lub komputerze PC. Nie jest szczególną niespodzianką, że wyszukiwanie słowa „komputer" na stronie *https://images.google.com/* powoduje wyświetlenie mnóstwa zdjęć tego rodzaju maszyn. Jednakże w bardziej ogólnym rozumieniu komputer może być dowolnym rodzajem urządzenia, które przechowuje i przetwarza dane.

Internetowa Encyklopedia PWN definiuje komputer jako „urządzenie elektroniczne przeznaczone do przetwarzania informacji (danych) przedstawionych w postaci cyfrowej, sterowane programem zapisanym w pamięci. Pojęcie komputera obejmuje obecnie zarówno komputery zaprogramowane na stałe, używane jako automaty sterujące, na przykład w urządzeniach gospodarstwa domowego, jak i komputery uniwersalne, dające się dowolnie zaprogramować"[1].

[1] Źródło: *https://encyklopedia.pwn.pl/haslo/komputer;3924650.html*.

Każdy typ komputera został zaprojektowany w unikatowy sposób, ale do budowy wszystkich urządzeń tego rodzaju zastosowano podobny **sprzęt** (ang. *hardware*). Dwa najważniejsze elementy sprzętowe to **procesor** (znany też jako CPU), którego zadaniem jest wykonywanie prostych obliczeń, oraz **pamięć operacyjna** (określana również jako RAM), która tymczasowo przechowuje dane. Na rysunku 1.1 przedstawiono wygląd obydwu tych komponentów.



Rysunek 1.1. Przykładowy procesor i układy pamięci operacyjnej

Użytkownicy zwykle oglądają i wykorzystują ekrany dotykowe, klawiatury i monitory, ale to procesory i pamięci są elementami, które wykonują właściwe operacje obliczeniowe. Obecnie panującym standardem jest zastosowanie w jednym urządzeniu co najmniej ośmiu procesorów i 4 gigabajtów (czyli czterech miliardów komórek) pamięci operacyjnej, nawet w przypadku zwykłego smartfona.

Czym jest programowanie?

Program jest sekwencją instrukcji, które określają, w jaki sposób należy przeprowadzić obliczenie przy użyciu sprzętu komputerowego. Obliczenie to może mieć charakter matematyczny, a więc może na przykład polegać na rozwiązaniu jakiegoś układu równań lub znalezieniu pierwiastków pewnego wielomianu. Mogłoby również stanowić obliczenie symboliczne, takie jak wyszukiwanie i zastępowanie tekstu w dokumencie lub też (o dziwo) kompilowanie programu.

Szczegóły różnią się tu w zależności od zastosowanego języka programowania, ale kilka podstawowych instrukcji występuje w niemal każdym z nich.

wejście:

Pobieranie danych z klawiatury, pliku, czujnika lub jakiegoś innego urządzenia.

wyjście:

Wyświetlanie danych na ekranie lub wysłanie danych do pliku albo innego urządzenia.

matematyka:

Przeprowadzanie podstawowych działań matematycznych, takich jak dodawanie i dzielenie.

decyzja:

Sprawdzanie występowania określonych warunków i wykonywanie odpowiedniego kodu.

powtarzanie:

Wielokrotne przeprowadzanie pewnych działań, zwykle z jakimiś zmianami.

Możesz w to wierzyć albo nie, ale w gruncie rzeczy właśnie do tego ogranicza się programowanie. Każdy program, z którego korzystałeś, nieważne jak byłby skomplikowany, składa się z niewielkich instrukcji, które wyglądają mniej więcej tak jak wymienione powyżej. Możesz zatem traktować **programowanie** jako proces dzielenia wielkich, złożonych zadań na coraz to mniejsze podzadania. Proces ten trwa aż do momentu, gdy podzadania są wystarczająco proste, aby mogły zostać wykonane za pomocą obwodów elektronicznych, z których składa się dany sprzęt komputerowy.

Program „Witaj, świecie!”

Tradycyjnie już pierwszy program, który się pisze, rozpoczynając naukę nowego języka programowania, nosi nazwę „programu »Witaj, świecie!«”. Wszystko, co robi ten program, ogranicza się do wyświetlenia słów „Witaj, świecie!” na ekranie komputera. W języku Java program ten może mieć następującą postać:

```
public class Hello {

   public static void main(String[] args) {
      // pokazanie na ekranie prostego komunikatu
      System.out.println("Witaj, świecie!");
   }
}
```

Uruchomienie tego programu skutkuje wyświetleniem na ekranie poniższego komunikatu:

```
Witaj, świecie!
```

Zwróć uwagę na fakt, że komunikat wyświetlony na ekranie nie zawiera znaków cudzysłowu.

Programy napisane w języku Java składają się z definicji *klas* i *metod*, a metody składają się z *instrukcji*. **Instrukcja** (ang. *statement*) to linia kodu, która wykonuje podstawową operację. W przedstawionym powyżej programie taką linią jest **instrukcja print**, która odpowiada za wyświetlenie komunikatu na ekranie:

```
System.out.println("Witaj, świecie!");
```

Instrukcja `System.out.println` wyświetla dane na ekranie; nazwa `println` to skrót od słów „wydrukuj linię” (ang. *print line*). Nieco mylące może być to, że angielskie słowo *print* może oznaczać zarówno czynność wyświetlania czegoś na ekranie, jak i wysyłania do drukarki. Z tego powodu w tej książce staramy się używać słowa „wyświetlanie”, gdy mamy na myśli pokazanie czegoś na ekranie. Podobnie jak większość instrukcji, również pokazana powyżej instrukcja wyświetlania jest zakończona znakiem średnika (`;`).

Java jest językiem, w którym rozróżnia się wielkość liter, co oznacza, że wielkie i małe litery nie są tym samym. W przedstawionym powyżej przykładzie słowo `System` musi się zaczynać wielką literą; `system` i `SYSTEM` się tu nie sprawdzą.

Metoda (ang. *method*) jest nazwaną sekwencją instrukcji. W naszym programie zdefiniowana jest jedna metoda, o nazwie `main`:

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

Nazwa i forma metody `main` są szczególne, ponieważ działanie programu rozpoczyna się od pierwszej należącej do niej instrukcji i kończy wraz z zakończeniem wykonywania ostatniej. W dalszej części książki będziemy mieli do czynienia z programami, w których definiowana będzie większa liczba metod.

W przedstawionym powyżej programie zdefiniowana została klasa o nazwie `Hello`. Na razie przyjmijmy, że **klasa** (ang. *class*) to zbiór metod; więcej informacji na ten temat poznasz wkrótce. Klasie można nadać dowolną nazwę, ale przyjęło się, że nazwa ta powinna się rozpoczynać wielką literą. Nazwa klasy musi pasować do nazwy pliku, w którym się ona znajduje, dlatego nasza klasa powinna zostać umieszczona w pliku o nazwie *Hello.java*.

W Javie używa się nawiasów klamrowych ({ oraz }) do grupowania instrukcji w pewną całość. W pliku *Hello.java* najbardziej zewnętrzne nawiasy klamrowe zawierają definicję naszej klasy, a wewnętrzne nawiasy klamrowe obejmują definicję metody.

Linia zaczynająca się dwoma ukośnikami (//) stanowi **komentarz** (ang. *comment*), będący napisanym w języku naturalnym (czyli na przykład po polsku czy po angielsku) fragmentem tekstu, który wyjaśnia działanie kodu. Gdy kompilator natrafia na znaki //, pomija wszystko, co występuje po nich aż do końca bieżącej linii. Komentarze nie mają żadnego wpływu na wykonanie programu, ale ułatwiają innymi programistom (jak również Tobie samemu) zrozumienie tego, co miałeś zamiar zrobić.

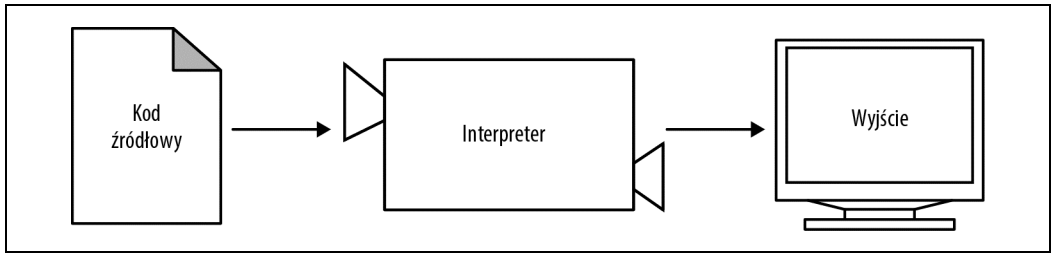
Kompilowanie programów w języku Java

Językiem programowania, którego naukę niebawem rozpoczniesz z tą książką, jest Java, będąca **językiem wysokiego poziomu** (ang. *high-level language*). Do innych wysokopoziomowych języków, o których mogłeś słyszeć, należą Python, C i C++, PH, Ruby oraz JavaScript.

Programy napisane w językach wysokiego poziomu, zanim będzie można je uruchomić, muszą zostać przetłumaczone na **język niskiego poziomu** (ang. *low-level language*), nazywany także „językiem maszynowym". Tłumaczenie to zajmuje trochę czasu, co stanowi pewną wadę języków wysokopoziomowych. Mają one jednak również dwie istotne zalety:

- *Znacznie* łatwiej jest programować przy użyciu języków wysokiego poziomu. Pisanie programów zajmuje w ich przypadku mniej czasu, są one też krótsze i łatwiej się je czyta, a także istnieje większa szansa, że będą poprawne.
- Programy napisane w językach wysokiego poziomu są **przenośne**, co oznacza, że mogą być uruchamiane na różnego rodzaju komputerach przy tylko niewielkich zmianach lub nawet kompletnym ich braku. Programy opracowane przy użyciu języków niskiego poziomu mogą działać wyłącznie na komputerach jednego typu.

Kod napisany w językach wysokopoziomowych jest tłumaczony na kod niskopoziomowy za pomocą dwóch rodzajów programów: interpreterów i kompilatorów (ang. *compiler*). **Interpreter** odczytuje program wysokiego poziomu i wykonuje go, co oznacza, że robi to, co nakazuje mu program. Przetwarza go kawałek po kawałku, na przemian odczytując poszczególne linie i przeprowadzając odpowiednie obliczenia. Na rysunku 1.2 została przedstawiona zasada działania interpretera.



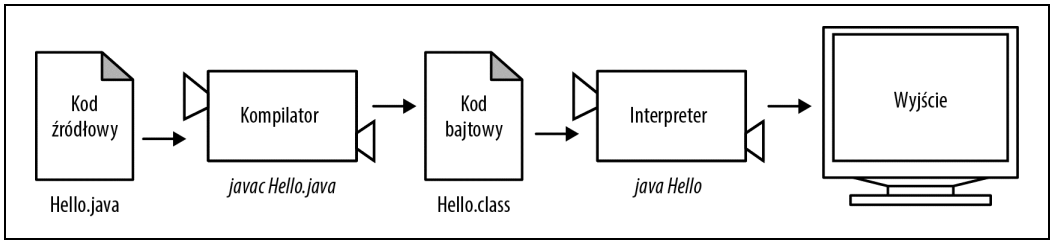
Rysunek 1.2. Sposób wykonywania kodu napisanego w językach interpretowanych

W odróżnieniu od tego **kompilator** odczytuje cały program na raz i tłumaczy go od początku do końca jeszcze przed rozpoczęciem wykonywania. W tym kontekście program napisany w języku wysokiego poziomu jest nazywany kodem źródłowym (ang. *source code*). Przetłumaczony program jest określany mianem **kodu obiektowego** (ang. *object code*) lub **wykonywalnego** (ang. *executable*). Po skompilowaniu programu można go uruchamiać wielokrotnie bez konieczności kolejnych translacji. Dzięki temu skompilowane programy zwykle działają znacznie szybciej niż interpretowane.

Warto odnotować, że kod obiektowy, jako zapisany w języku niskiego poziomu, nie jest przenośny. Nie da się uruchomić pliku wykonywalnego skompilowanego na laptop działający pod kontrolą systemu operacyjnego Windows na przykład na telefonie z systemem Android. Aby program można było uruchamiać na maszynach różnych typów, trzeba go skompilować wielokrotnie. Pisanie kodu źródłowego, który da się poprawnie kompilować i uruchamiać na różnego rodzaju komputerach, może być dość trudne.

Java, aby mogła sprostać temu wyzwaniu, jest językiem *zarówno* kompilowanym, *jak i* interpretowanym. Zamiast tłumaczyć kod źródłowy bezpośrednio do postaci wykonywalnej, kompilator Javy generuje kod dla **maszyny wirtualnej** (ang. *virtual machine*). Ta „wyimaginowana" maszyna zapewnia możliwości, które są wspólne dla komputerów biurkowych, laptopów, tabletów, telefonów i tak dalej. Wykorzystywany przez nią język, nazywany **kodem bajtowym** (ang. *byte code*) Javy, przypomina kod obiektowy i można go łatwo i szybko interpretować.

Dzięki temu program napisany w Javie da się skompilować na jednej maszynie, przenieść jego kod bajtowy na inną, a następnie go na niej uruchomić. Na rysunku 1.3 zostały przedstawione kroki tego procesu przekształcania. Kompilator języka Java to program o nazwie `javac`. Tłumaczy on pliki o rozszerzeniu `java` na pliki `class`, które przechowują otrzymany kod bajtowy. Interpreter Javy to z kolei program, który nosi nazwę `java`, będącą skrótem terminu „maszyna wirtualna Javy" (ang. *Java Virtual Machine*, *JVM*).



Rysunek 1.3. Proces kompilowania i uruchamiania programu napisanego w języku Java

Programista pisze kod źródłowy stanowiący zawartość pliku *Hello.java*, a następnie używa programu `javac`, aby skompilować ten plik. Jeśli nie ma w nim żadnych błędów, kompilator zapisuje kod bajtowy w pliku *Hello.class*. Aby uruchomić swój program, programista korzysta z narzędzia `java` w celu rozpoczęcia interpretowania wynikowego kodu bajtowego. Rezultat działania programu jest następnie wyświetlany na ekranie komputera.

Choć może się to wydawać dość skomplikowane, wykonanie tych kroków jest w pełni zautomatyzowane w przypadku większości środowisk programistycznych. Dzięki temu, aby skompilować i zinterpretować swój program, zwykle musisz w nich tylko nacisnąć odpowiedni przycisk lub wpisać pojedyncze polecenie. Z drugiej strony warto wiedzieć, co dzieje się w tle, aby w sytuacji, gdy coś pójdzie nie tak, można było stwierdzić, co to było.

Wyświetlanie dwóch komunikatów

W metodzie `main` możesz umieścić dowolną liczbę instrukcji. Aby na przykład wyświetlić na ekranie komputera więcej niż jeden wiersz tekstu, możesz napisać następujący program:

```
public class Hello2 {

   public static void main(String[] args) {
      // pokazanie na ekranie prostego komunikatu
      System.out.println("Witaj, świecie!");    // pierwsza linia
      System.out.println("Jak się masz?");      // druga linia
   }
}
```

Jak również widać w powyższym przykładzie, komentarze można umieszczać zarówno na końcach linii kodu, jak i w zupełnie osobnych wierszach.

Frazy znajdujące się pomiędzy znakami cudzysłowu noszą nazwę **łańcuchów** lub **ciągów znakowych** (ang. *strings*), ponieważ zawierają sekwencje znaków połączonych ze sobą w pamięci. Znakami tymi mogą być litery, cyfry, znaki przestankowe, symbole, spacje, tabulacje i tak dalej.

Instrukcja `System.out.println` dodaje na końcu wiersza tekstu specjalny **znak nowej linii**, który przenosi kursor na początek następnego wiersza na ekranie. Jeśli nie chcesz, żeby tak się działo, zamiast `println` możesz zastosować `print`, jak zostało to pokazane poniżej:

```
public class Goodbye {

   public static void main(String[] args) {
      System.out.print("Żegnaj, ");
```

```
        System.out.println("okrutny świecie!");
    }
}
```

W tym przykładzie pierwsza instrukcja nie dodaje znaku nowej linii, dzięki czemu widoczny na ekranie tekst ma postać:

```
Żegnaj, okrutny świecie!
```

Zauważ, że na końcu pierwszego łańcucha znajduje się spacja, którą widać też na ekranie tuż przed słowem `okrutny`.

Formatowanie kodu źródłowego

W kodzie źródłowym napisanym w języku Java niektóre spacje są wymagane. Spacjami muszą być na przykład rozdzielone poszczególne słowa kluczowe i nazwy, dlatego poniższy program jest niepoprawny:

```
publicclassGoodbye {

   publicstaticvoidmain(String[] args) {
      System.out.print("Żegnaj, ");
      System.out.println("okrutny świecie!");
   }
}
```

Jednak większość pozostałych spacji jest opcjonalna. Na przykład przedstawiony poniżej program *jest* zupełnie poprawny:

```
public class Goodbye {
public static void main(String[] args) {
System.out.print("Żegnaj, ");
System.out.println("okrutny świecie!");
}
}
```

Znaki nowej linii są opcjonalne. Moglibyśmy zatem po prostu napisać:

```
public class Goodbye { public static void main(String[] args) { System.out.print("Żegnaj, ");
➥System.out.println("okrutny świecie!");}}
```

Program będzie nadal działał, staje się jednak znacznie mniej czytelny. Znaki nowej linii i spacje bardzo się przydają przy wizualnej organizacji kodu, gdyż sprawiają, że łatwiej będzie zrozumieć program i znaleźć w nim błędy, gdy się pojawią.

Wiele narzędzi do edycji kodu automatycznie formatuje kod źródłowy przy użyciu spójnego systemu wcięć i podziałów linii. Na przykład korzystając z rozwiązania DrJava (o którym więcej w podrozdziale „Instalowanie programu DrJava”), możesz zastosować wcięcie dla całego zaznaczonego kodu. W tym celu powinieneś użyć skrótu klawiszowego *Ctrl+A* (aby zaznaczyć cały tekst), a następnie nacisnąć klawisz *Tab* (aby dodać wcięcie).

Organizacje intensywnie zajmujące się pisaniem programów komputerowych zwykle stosują bardzo rygorystyczne wytyczne dotyczące tego, jak należy formatować kod źródłowy. Firma Google

publikuje na przykład swoje standardy programowania w Javie, które powinny być przestrzegane w projektach o otwartym kodzie. Informacje na ten temat znajdziesz pod adresem: *http://google.github.io/styleguide/javaguide.html*.

Na tym etapie prawdopodobnie jeszcze nie będziesz rozumieć tych wytycznych, ponieważ odnoszą się one do konstrukcji języka, których nie miałeś okazji poznać. Regularne odwoływanie się do nich w trakcie lektury tej książki może się jednak okazać bardzo pomocne.

Używanie sekwencji ucieczki

Możliwe jest wyświetlanie na ekranie komputera wielu linii tekstu za pomocą tylko jednego wiersza kodu. W takim przypadku musisz jedynie poinformować Javę, gdzie mają się znaleźć znaki podziału linii.

```
public class Hello3 {

   public static void main(String[] args) {
      System.out.print("Witaj!\nJak się masz?\n");
   }
}
```

Tekst widoczny na ekranie składa się z dwóch linii, z których każda kończy się znakiem nowego wiersza:

```
Witaj!
Jak się masz?
```

Każdy ciąg `\n` to **sekwencja ucieczki** (ang. *escape sequence*) lub dwa znaki kodu źródłowego, które reprezentują jeden znak (lewy ukośnik umożliwia „ucieczkę" od dosłownej interpretacji literałów łańcuchowych). Zauważ, że pomiędzy znakami `\n` a słowem `Jak` nie występuje spacja. Jeśli ją tam umieścisz, druga linia będzie się zaczynała od odstępu. W Javie dostępnych jest w sumie osiem sekwencji ucieczki, a cztery najczęściej używane zostały przedstawione w tabeli 1.1.

Tabela 1.1. Często wykorzystywane sekwencje ucieczki

Sekwencja	Znaczenie
`\n`	nowa linia
`\t`	tabulacja
`\"`	cudzysłów
`\\`	lewy ukośnik

Na przykład, aby umieścić w łańcuchu znakowym znaki cudzysłowu, musisz zastosować odpowiednią sekwencję ucieczki, poprzedzając każdy cudzysłów lewym ukośnikiem, jak zostało to pokazane poniżej:

```
System.out.println("Powiedziała mi \"Cześć!\".");
```

Wynikiem tego będzie wyświetlenie na ekranie następującej linii tekstu:

```
Powiedziała mi "Cześć!".
```

Czym jest informatyka?

W książce tej celowo pomijamy niektóre szczegóły dotyczące języka Java (takie jak pozostałe sekwencje ucieczki), ponieważ głównym celem, który nam przyświeca, jest nauczenie Cię sposobu myślenia, jakim posługują się zawodowi informatycy. Umiejętność zrozumienia zasad działania programów jest znacznie cenniejsza niż sama umiejętność pisania kodu.

Jeśli chciałbyś dowiedzieć się więcej na temat samego języka Java, wiedz, że firma Oracle udostępnia zestaw oficjalnych materiałów szkoleniowych na swojej stronie internetowej, pod adresem: *https://docs.oracle.com/javase/tutorial/*. Lekcja zatytułowana „Language Basics" (Podstawy języka), dostępna w dziale „Learning the Java Language" (Nauka języka Java), to coś, od czego warto tu zacząć.

Jednym z najciekawszych aspektów pisania programów jest decydowanie o tym, w jaki sposób rozwiązać określony problem, zwłaszcza gdy istnieje wiele różnych rozwiązań. Dostępnych jest na przykład mnóstwo metod porządkowania zbiorów liczb, a każda z nich ma pewne zalety. Aby określić, która z nich sprawdzi się najlepiej w danej sytuacji, potrzebne są nam techniki umożliwiające formalne opisywanie i analizowanie rozwiązań.

Algorytm to sekwencja kroków, które określają, jak rozwiązać dany problem. Pewne algorytmy są szybsze niż inne, a niektóre wymagają użycia mniejszej ilości miejsca w pamięci komputera. **Informatyka** to nauka o algorytmach, zajmująca się również ich odkrywaniem i analizą. Wraz z tym, jak uczysz się opracowywać algorytmy rozwiązujące problemy, z którymi nie miałeś wcześniej do czynienia, uczysz się również myśleć jak informatyk.

Projektowanie algorytmów i pisanie kodu to zadania trudne i podatne na błędy. Ze względów historycznych błędy programistyczne są określane mianem **bugów** (ang. *bug* — robak, pluskwa), proces ich tropienia i poprawiania nosi zaś nazwę **debugowania** (ang. *debugging* — odrobaczanie, odpluskwianie). Ucząc się debugować swoje programy, będziesz rozwijał umiejętności rozwiązywania problemów. Będziesz musiał myśleć kreatywnie, gdy pojawią się niespodziewane błędy.

Choć może to być naprawdę frustrujące zajęcie, debugowanie to niezwykle bogata intelektualnie, wymagająca i interesująca część programowania komputerowego. Przypomina ono często pracę detektywa. Masz tu do czynienia z pewnymi poszlakami i musisz wywnioskować, jakie procesy i zdarzenia powodują pojawienie się obserwowanych rezultatów. Rozmyślania na temat tego, jak poprawiać programy i zwiększać wydajność ich działania, prowadzą nawet niekiedy do odkrycia zupełnie nowych algorytmów.

Debugowanie programów

Dobrym pomysłem jest czytanie tej książki przed komputerem, abyś mógł wypróbowywać poszczególne przykłady w trakcie lektury. Wiele z tych przykładów możesz uruchamiać bezpośrednio z poziomu panelu *Interactions* narzędzia DrJava (więcej na ten temat w podrozdziale zatytułowanym „Panel Interactions programu DrJava"). Gdy jednak umieścisz kod w pliku źródłowym, łatwiej będzie wypróbować różne warianty.

Zawsze gdy eksperymentujesz z nową funkcją, powinieneś również próbować popełniać błędy. Sprawdź na przykład, co się stanie z programem „Witaj, świecie!", gdy z jego kodu pozbędziesz się jednego ze znaków cudzysłowu. A co będzie, gdy usuniesz obydwa? Przekonaj się, jaki skutek będzie miało niepoprawne napisanie nazwy `println`. Tego rodzaju eksperymenty pomagają zapamiętać to, co czytasz. Pomagają też w debugowaniu kodu, ponieważ dzięki nim uczysz się znaczenia poszczególnych komunikatów o błędach. Lepiej popełniać tego rodzaju pomyłki teraz i celowo niż później i przypadkiem.

Debugowanie jest jak nauka eksperymentalna — gdy masz już jakiś pomysł dotyczący tego, co jest nie tak, modyfikujesz swój program i próbujesz ponownie. Jeśli Twoja hipoteza była poprawna, wówczas możesz przewidzieć wynik wprowadzonej zmiany i jesteś o krok bliżej celu, którym jest działający program. Jeśli Twoja hipoteza była niepoprawna, musisz postarać się o nową.

Programowanie i debugowanie powinny iść ręka w rękę. Staraj się nie pisać mnóstwa kodu, a następnie przeprowadzać debugowania metodą prób i błędów aż do czasu, gdy wszystko zaczyna prawidłowo działać. Zamiast tego zacznij od programu, który *coś* robi, i dokonuj niewielkich modyfikacji, debugując go w międzyczasie, aż zacznie robić dokładnie to, co chciałeś. Dzięki temu zawsze będziesz miał do dyspozycji działający program, a ponadto łatwiej będzie Ci wyizolować błędy.

Doskonałym przykładem zastosowania tego podejścia jest system operacyjny Linux, który składa się z milionów linii kodu. Zaczął on swoje istnienie jako prosty program, którego Linus Torvalds używał w celu zbadania możliwości układu 80836 firmy Intel. Jak napisał Larry Greenfield w swoim *The Linux Users' Guide*: „Jednym z wcześniejszych projektów Linusa był program, który wyświetlał na przemian ciągi AAAA i BBBB. Z czasem wyewoluował on w Linuksa".

Wreszcie, programowanie wyzwala czasem silne emocje. Gdy zmagasz się z trudnym bugiem, możesz czuć złość, przygnębienie lub zakłopotanie. Pamiętaj, że nie jesteś w tym osamotniony i właściwie każdy programista ma podobne doświadczenia. Nie wahaj się zwrócić do kolegi i zadać pytanie!

Słownictwo

W całej tej książce staramy się definiować każdy termin za pierwszym razem, gdy go używamy. Na końcu każdego rozdziału zamieszczamy nowe pojęcia i ich definicje w kolejności, w jakiej pojawiają się one w treści. Jeśli poświęcisz trochę czasu na naukę tego słownictwa, łatwiej będzie Ci czytać kolejne rozdziały.

rozwiązywanie problemu:

Proces formułowania problemu, znajdowania rozwiązania i wyrażania tego rozwiązania.

sprzęt komputerowy:

Komponenty elektroniczne i mechaniczne, z których składa się komputer; mogą to być na przykład: procesory, pamięci operacyjne i dyski twarde.

procesor:

Czip komputerowy wykonujący proste operacje, na przykład podstawowe działania arytmetyczne i logiczne.

pamięć operacyjna:

Obwody przechowujące dane tak długo, jak długo włączony jest komputer. Nie należy ich mylić z trwałymi nośnikami danych, takimi jak dyski twarde czy nośniki flash.

program:

Sekwencja instrukcji, które określają, jak należy wykonywać zadania za pomocą komputera. Bywa też określany mianem „oprogramowania".

programowanie:

Zastosowanie technik rozwiązywania problemów do opracowania wykonywalnych programów komputerowych.

instrukcja:

Część programu opisująca jeden krok algorytmu.

instrukcja print:

Instrukcja, której wykonanie powoduje wyświetlenie danych na ekranie.

metoda:

Nazwana sekwencja instrukcji.

klasa:

Na razie: zbiór związanych ze sobą metod (w dalszej części książki dowiesz się, że to jeszcze nie wszystko).

komentarz:

Część programu, która zawiera informacje na jego temat, ale nie ma wpływu na to, jak działa.

język wysokiego poziomu:

Język programowania, który został opracowany w taki sposób, aby ludzie mogli go łatwo czytać i pisać w nim.

język niskiego poziomu:

Język programowania, który został opracowany w taki sposób, aby komputer mógł łatwo uruchamiać napisany w nim program. Bywa również nazywany „językiem maszynowym" lub „językiem asemblera".

przenośność:

Możliwość uruchamiania programu na więcej niż jednym rodzaju komputera.

interpretacja:

Uruchamianie programu napisanego w języku wysokiego poziomu przez translację jego pojedynczych linii i natychmiastowe wykonywanie odpowiednich instrukcji.

kompilacja:

Translacja całego programu napisanego w języku wysokiego poziomu na język niskiego poziomu w celu przygotowania go do późniejszego wykonania.

kod źródłowy:

Program napisany w języku wysokiego poziomu przed poddaniem go kompilacji.

kod obiektowy:

Wynik działania kompilatora po translacji programu.

kod wykonywalny:

Inna nazwa kodu obiektowego, który jest gotowy do uruchomienia na określonym sprzęcie.

maszyna wirtualna:

Emulacja prawdziwej maszyny. JVM umożliwia uruchamianie na komputerze programów napisanych w języku Java.

kod bajtowy:

Szczególny rodzaj kodu obiektowego wykorzystywany w przypadku programów napisanych w języku Java. Kod bajtowy przypomina nieco kod obiektowy, ale jest przenośny, podobnie jak kod napisany w języku wysokopoziomowym.

łańcuch znakowy:

Sekwencja znaków; podstawowy typ danych do przechowywania tekstu.

znak nowej linii:

Specjalny znak oznaczający koniec linii tekstu. Znany również jako zakończenie linii, znak końca linii (ang. *end of line*, *EOL*) lub podział linii.

sekwencja ucieczki:

Sekwencja kodu reprezentująca znak specjalny w obrębie łańcucha znakowego.

algorytm:

Procedura lub formuła umożliwiająca rozwiązanie problemu przy użyciu komputera lub bez niego.

informatyka:

Naukowe i praktyczne podejście do techniki obliczeniowej i jej zastosowania.

bug:

Błąd w programie.

debugowanie:

Proces odnajdywania i poprawiania błędów w programie.

Ćwiczenia

Na końcu każdego rozdziału zamieszczamy ćwiczenia dotyczące omówionego materiału, które możesz samodzielnie wykonać, aby utrwalić zdobytą wiedzę. Zachęcamy Cię do podjęcia przynajmniej próby zmierzenia się z każdym z przedstawionych tu problemów. Nie da się bowiem nauczyć programować, wyłącznie czytając teksty na temat programowania; trzeba również praktykować.

Zanim będziesz mógł kompilować i uruchamiać programy napisane w języku Java, będziesz musiał pobrać i zainstalować kilka niezbędnych narzędzi. Wybór jest tu dość szeroki, my jednak rekomendujemy Ci program DrJava, będący „zintegrowanym środowiskiem programisty" (ang. *integrated development environment*, *IDE*), świetnie dostosowanym do potrzeb osób początkujących. Wskazówki dotyczące rozpoczęcia pracy z tym narzędziem znajdziesz w podrozdziale „Instalowanie programu DrJava" należącym do dodatku A.

Kod związany z tym rozdziałem znajdziesz w katalogu *r01* archiwum kodów dołączonego do niniejszej książki. Informacje na temat tego, jak pobrać to archiwum, znajdziesz w należącym do wstępu podrozdziale „Używanie przykładowych kodów". Zanim zaczniesz wykonywać przedstawione poniżej ćwiczenia, radzimy Ci skompilować i uruchomić przykłady zaprezentowane w tym rozdziale.

Ćwiczenie 1.1

Informatycy mają denerwujący zwyczaj używania zwykłych angielskich słów do określania rzeczy, które w codziennym języku angielskim mają zupełnie inne znaczenie. Na przykład słowa instrukcja (ang. *statement*) i komentarz (ang. *comment*) po angielsku oznaczają zasadniczo to samo (stwierdzenie), jednak w przypadku programowania ich znaczenia są oczywiście zupełnie różne.

1. Jaka w żargonie komputerowym jest różnica znaczeniowa pomiędzy instrukcją a komentarzem?
2. Co znaczy, że program jest przenośny?
3. Co w języku naturalnym oznacza słowo „kompilować"?
4. Co to jest kod wykonywalny?

Słowniczek zamieszczony na końcu każdego rozdziału ma na celu wyróżnienie i przypomnienie słów oraz zwrotów, które mają specjalne znaczenie w informatyce. Gdy widzisz znajome słowa w tej dziedzinie, nie zakładaj, że wiesz, co znaczą!

Ćwiczenie 1.2

Zanim zrobisz coś innego, dowiedz się, jak skompilować i uruchomić program napisany w Javie. Niektóre środowiska zapewniają przykładowe programy podobne do przykładu, który zamieściliśmy w podrozdziale „Program »Witaj, świecie!«" należącym do tego rozdziału.

1. Przepisz program „Witaj, świecie!", a następnie skompiluj go i uruchom.
2. Dodaj instrukcję `print`, która będzie wyświetlała drugi komunikat po tekście „Witaj, świecie!". Niech będzie to coś błyskotliwego, w stylu: „Jak się masz?". Ponownie skompiluj i uruchom swój program.
3. Dodaj do programu jakiś komentarz (gdziekolwiek), przekompiluj go i uruchom ponownie. Nowy komentarz nie powinien mieć żadnego wpływu na wynik działania programu.

Ćwiczenie to może Ci się wydawać trywialne, stanowi jednak punkt wyjścia dla wielu innych programów, z którymi będziemy mieć do czynienia w dalszej części książki. Aby móc swobodnie debugować swoje przyszłe programy, musisz się oswoić z używanym środowiskiem programistycznym.

W niektórych środowiskach łatwo stracić kontrolę nad tym, który program jest w danej chwili wykonywany. Może się okazać, że będziesz próbował debugować jeden program w czasie, gdy przez przypadek uruchomiony będzie zupełnie inny. Dodawanie (i zmienianie) instrukcji `print` to prosty sposób, aby się upewnić, że program, na który patrzysz, jest tym, który uruchomiłeś.

Ćwiczenie 1.3

Na tym etapie dobrze jest popełniać tak wiele błędów, jak tylko zdołasz wymyślić, abyś mógł zapoznać się z komunikatami o błędzie, które zgłasza kompilator. Czasami zdarza się, że informuje Cię on o dokładnej przyczynie problemu, a Twoim zadaniem jest po prostu jej wyeliminowanie. Nieraz jednak komunikaty o błędzie wprowadzają sporo zamieszania. Z czasem rozwiniesz w sobie zmysł informujący Cię, kiedy można ufać kompilatorowi, a kiedy należy samemu dowiedzieć się, gdzie leży problem.

Zaczynając od programu „Witaj, świecie!", spróbuj wprowadzić wymienione poniżej błędy. Następnie skompiluj program, przeczytaj komunikat o błędzie (jeśli taki się pojawi), po czym napraw błąd.

1. Usuń jeden z otwierających nawiasów klamrowych.
2. Usuń jeden z zamykających nawiasów klamrowych.
3. Zamiast `main` napisz `mian`.
4. Usuń słowo `static`.
5. Usuń słowo `public`.
6. Usuń słowo `System`.
7. Zastąp słowo `println` słowem `Println`.
8. Zastąp słowo `println` słowem `print`.
9. Usuń jeden z nawiasów. Dopisz jeden dodatkowy.

Myśl jak prawdziwy informatyk, koduj jak profesjonalny programista!

Java jest językiem dojrzałym i jednocześnie bardzo nowoczesnym. Skupiona wokół niego społeczność cały czas dynamicznie go rozwija, sprawiając, że wszechstronność i innowacyjność Javy budzi podziw. Jest to też język idealny do nauki programowania — początkujący programiści, którzy wybierają Javę jako swój pierwszy język, w naturalny sposób nabierają dobrych nawyków, dzięki czemu później stosowanie się do najlepszych praktyk programistycznych nie sprawia im problemu. Ważne jest tylko, aby wraz z nauką programowania adept sztuki tworzenia kodu przyswoił choćby najważniejsze prawidła informatyki. Pozwala to uniknąć w przyszłości wielu kłopotów z niewydajnym, niezrozumiałym i trudnym do utrzymania kodem.

Oto zaktualizowane i uzupełnione wydanie znakomitego podręcznika dla początkujących, dzięki któremu zdobędziesz solidne podstawy informatyki i programowania w Javie. Wyjaśniono tu szereg skomplikowanych tematów, rozłożonych na mniejsze zagadnienia, z których każde zostało opatrzone zrozumiałymi przykładami. Książka zawiera mnóstwo ćwiczeń, które sprawią, że zaczniesz kreatywnie podchodzić do programowania, a odkrywanie relacji między danymi wejścia i wyjścia, klasami, metodami i obiektami przyniesie Ci prawdziwą satysfakcję. Duży nacisk położono na właściwe słownictwo i... właśnie tworzenie programów. Zapoznasz się więc z różnymi strategiami projektowania, pisania, testowania i debugowania programów. Liczne przykłady kodu, ćwiczenia, podsumowania, porady i wskazówki stanowią wspaniałe uzupełnienie prezentowanych treści.

W tej książce między innymi:

- podstawowe koncepcje programowania
- zmienne, wartości, zarządzanie pamięcią, operacje wejścia-wyjścia
- operacje logiczne, pętle i referencje
- obiekty, tablice, klasy i ich projektowanie
- konstruktory, interfejsy, detektory zdarzeń

Dr Allen B. Downey jest profesorem informatyki na Olin College of Engineering; wcześniej uczył tego przedmiotu między innymi na Uniwersytecie Kalifornijskim w Berkeley. Jest autorem kilku książek z serii *Myśl w języku...*, w tym *Myśl w języku Python*.

Dr Chris Mayfield jest adiunktem informatyki na James Madison University, gdzie zajmuje się badaniami nad edukacją informatyczną oraz rozwojem zawodowym.

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

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-6719-7

9 788328 367197

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł