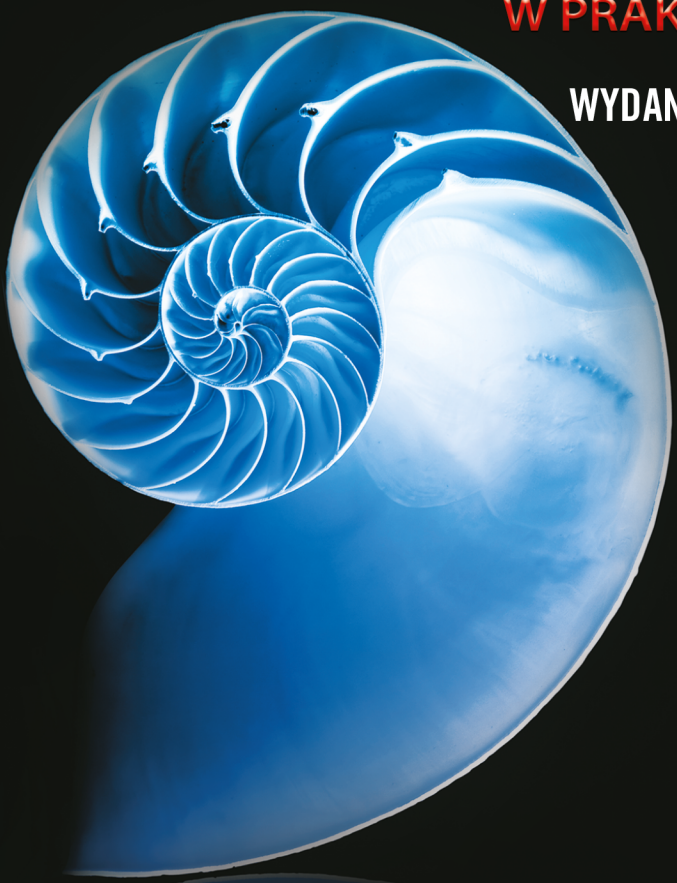
PROGRAMOWANIE w Javie

SOLIDNA WIEDZA W PRAKTYCE

WYDANIE XI



PAUL DEITEL | HARVEY DEITEL

Helion

Tytuł oryginału: Java How to Program, Early Objects (11th Edition)

Tłumaczenie: Rafał Jońca

ISBN: 978-83-283-3973-6

Authorized translation from the English language edition, entitled: JAVA HOW TO PROGRAM, EARLY OBJECTS, Eleventh Edition; ISBN 0134743350; by Harvey M. Deitel; and by Paul J. Deitel; published by Pearson Education, Inc. Copyright © 2018, 2015, 2012 and 2009 by Pearson Education, Inc., Hoboken, New Jersey 07030.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

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

Printed in Poland.

Spis treści

3. Wprowadzenie do klas, obiektów, metod i tekstów 127

Wprowadzenie do aplikacji Javy, wejścia – wyjścia i operatorów

2

Cele

W tym rozdziale:

- Proste aplikacje Javy
- Instrukcje wejścia i wyjścia
- Typy podstawowe Javy
- Podstawowe koncepcje dotyczące pamięci
- Operatory arytmetyczne
- Kolejność wykonywania działań arytmetycznych
- Instrukcje podejmujące decyzje
- Operatory relacji i równości

Plan rozdziału

2.1. Wprowadzenie

Ten rozdział stanowi wprowadzenie do programowania w języku Java. Zaczynamy od przykładów, które wyświetlą na ekranie prosty tekst. Następnie przejdziemy do programu, który pobiera od użytkownika dwie wartości, wylicza ich sumę i wyświetla wynik. Dowiesz się, jak poinstruować komputer, aby wykonywał operacje arytmetyczne i zapisywał wyniki w celu późniejszego ich użycia. Ostatni przykład pokaże, jak podejmować decyzje. Aplikacja porówna dwie liczby i wyświetli komunikat informujący o wyniku porównania. Do kompilacji i uruchamiania programów użyjemy narzędzi wiersza poleceń z JDK. Jeśli chcesz skorzystać z IDE, materiały wprowadzające znajdziesz pod adresem:

http://www.deitel.com/books/jhtp11

Zawarte są tam informacje o użyciu trzech najpopularniejszych IDE: Eclipse, NetBeans i IntelliJ IDEA.

2.2. Twój pierwszy program — wyświetlenie wiersza tekstu

Aplikacja Javy to program komputerowy, który wykona pewne działania, gdy użyjemy polecenia `java` do uruchomienia maszyny wirtualnej Javy. Punkty 2.2.1 i 2.2.2 zawierają informacje na temat kompilacji i uruchomienia aplikacji Javy. Najpierw zajmijmy się bardzo prostą aplikacją, która wyświetla jeden wiersz tekstu. Rysunek 2.1 przedstawia kod programu i wynik jego działania.

Będziemy wykorzystywać numery wierszy, aby ułatwić odniesienie się do konkretnego wiersza kodu — **nie** stanowią one części programu Javy. Ten prosty przykład ilustruje kilka istotnych elementów Javy. Wiersz 7. wykonuje całe zadanie — wyświetla na ekranie tekst `Witamy programistę Javy!`.

Umieszczanie komentarzy w programach

Wstawiamy komentarze w kodzie programu, aby go udokumentować i poprawić czytelność. Kompilator Javy **ignoruje** komentarze, więc **nie** powodują one żadnej akcji ze strony komputera wykonującego program.

```
1 // Rysunek 2.1. Welcome1.java
2 // Program wyświetlający tekst
3
4 public class Welcome1 {
5    // Metoda main rozpoczyna wykonywanie aplikacji Javy
6    public static void main(String[] args) {
7       System.out.println("Witamy programistę Javy!");
8    } // Koniec metody main
9 } // Koniec klasy Welcome1
```

```
Witamy programistę Javy!
```

Rysunek 2.1. Program wyświetlający tekst

Stosujemy konwencję, w której każdy program zaczynamy od komentarza wskazującego numer rysunku i nazwę pliku z programem. Komentarz z pierwszego wiersza to:

```
// Rysunek 2.1. Welcome1.java
```

Zaczyna się od znaków //, co wskazuje na **komentarz jednowierszowy**, który kończy się wraz z końcem wiersza zawierającego znaki //. Komentarz jednowierszowy nie musi zaczynać się od początku wiersza, lecz może zaczynać się w jego środku, jak ma to miejsce w wierszach 5., 8. i 9. Wiersz 2.:

```
// Program wyświetlający tekst
```

to dalsza część stosowanej konwencji. Wyjaśniamy tutaj działanie programu.

Java wykorzystuje też **komentarze tradycyjne**, które swym zasięgiem mogą obejmować kilka wierszy kodu:

```
/* To komentarz tradycyjny,
   który jest podzielony na kilka wierszy */
```

Taki komentarz rozpoczyna się od znaków /* i kończy znakami */. Kompilator ignoruje cały tekst znajdujący się między tymi znakami. Oba przedstawione rodzaje komentarzy Java odziedziczyła po językach C (komentarze tradycyjne) i C++ (komentarze jednowierszowe).

Java obsługuje również komentarze trzeciego typu — **komentarze Javadoc**. Są umieszczane między znakami /** i */. Kompilator ignoruje cały tekst między nimi. Komentarze Javadoc umożliwiają osadzanie dokumentacji programu bezpośrednio w kodzie źródłowym. Stanowią one zalecany sposób dokumentowania kodu dla innych programistów. **Program narzędziowy `javadoc`** (część JDK) odczytuje komentarze Javadoc i na ich podstawie generuje dokumentację w formacie stron HTML5. Aby oszczędzić miejsce, stosujemy w przykładach komentarze ze znakami // zamiast komentarzy tradycyjnych lub Javadoc.

Wykorzystanie pustych wierszy

Puste wiersze (takie jak wiersz 3.) oraz znaki spacji i tabulacji zwiększają czytelność programu. Wszystkie te elementy nazywa się zbiorczo **białymi spacjami**. Kompilator ignoruje białe spacje.

2.1. Typowy błąd programistyczny

Brak jednego z elementów definiujących komentarz tradycyjny lub Javadoc spowoduje zgłoszenie błędu składniowego. ***Błąd składniowy*** *to błąd zgłaszany przez kompilator, gdy napotka on fragment kodu łamiący zasady języka Java (np. jego składnię). Zasady składni Javy przypominają zasady gramatyczne języka naturalnego, np. języka polskiego, francuskiego czy angielskiego. Błędy składniowe nazywane są również* ***błędami kompilacji*** *lub* ***błędami kompilatora****, ponieważ kompilator wykrywa je w trakcie kompilacji programu. Gdy zostanie odkryty błąd składniowy, kompilator wyświetla stosowny komunikat. Aby pomyślnie skompilować program, musisz usunąć wszystkie błędy składniowe.*

2.1. Dobra praktyka programistyczna

Niektóre firmy i organizacje wymagają, aby każdy program rozpoczynał się od komentarza wskazującego cel powstania pliku, jego autora, a także datę i czas ostatniej modyfikacji.

2.2. Dobra praktyka programistyczna

Korzystaj z białych spacji, aby zwiększyć czytelność kodu.

Deklaracja klasy

Wiersz 4.:

```
public class Welcome1 {
```

rozpoczyna się od **deklaracji klasy** o nazwie `Welcome1`. Każdy program Javy składa się z przynajmniej jednej klasy zdefiniowanej przez programistę. **Słowo kluczowe `class`** informuje o rozpoczęciu deklaracji klasy, a tuż za nim pojawia się **nazwa klasy** (`Welcome1`). Słowa kluczowe w Javie to słowa zarezerwowane przez język i są zawsze pisane małymi literami. Kompletną listę wszystkich słów kluczowych Javy znajdziesz w dodatku C.

W rozdziałach od 2. do 7. wszystkie definiowanie klasy będą rozpoczynały się słowem kluczowym **`public`**. Na razie po prostu go potrzebujemy. Więcej informacji na temat klas publicznych (`public`) i niepublicznych znajdziesz w rozdziale 8.

Nazwa pliku dla klasy publicznej

Klasa publiczna **musi** być umieszczona w pliku o nazwie *NazwaKlasy.java*, więc klasa `Welcome1` znajduje się w pliku *Welcome1.java*.

2.2. Typowy błąd programistyczny

Jeśli nazwa pliku klasy publicznej nie jest równa nazwie klasy i rozszerzeniu .java*, kompilator zgłosi błąd kompilacji. Pamiętaj, aby użyć właściwej wielkości liter.*

Nazwy klas i identyfikatory

Zgodnie z konwencją nazwy klas rozpoczyna się od wielkiej litery, a następnie stosuje wielką literę dla pierwszej litery każdego z następnych słów (np. `ToJestNazwa` ↪`Klasy`). Nazwa klasy jest identyfikatorem — ciągiem znaków składającym się z liter, cyfr, podkreśleń (_) i znaków dolara ($), ale **nie** może zaczyna się od cyfry

ani zawierać znaków spacji. Oto kilka przykładów poprawnych identyfikatorów: `Witaj1`, `$wartosc`, `_wartosc`, `m_poleNumeru1` i `przycisk7`. Nazwa `7przycisk` **nie** jest poprawnym identyfikatorem, ponieważ rozpoczyna się od cyfry, a `pole wejsciowe` **nie** jest poprawnym identyfikatorem, bo zawiera znak spacji. Zazwyczaj identyfikator, który nie zaczyna się od wielkiej litery, nie jest nazwą klasy. W Javie istotna jest wielkość liter — małe i wielkie litery się różnią — więc `test` i `Test` to dwa różne (i całkowicie poprawne) identyfikatory.

2.3. Dobra praktyka programistyczna

Stosuje się konwencję, w której każdy wyraz identyfikatora nazwy klasy pisany jest z wielkiej litery. Identyfikator klasy `DolarTestowy` zaczyna się od słowa `Dolar` z wielką literą `D` i stosuje drugie słowo `Testowy` z wielką literą `T`. Ten sposób nazewnictwa nazywa się ***stylem wielbłądzim****, ponieważ wielkie litery przypominają garby wielbłąda.*

Podkreślenie (_) w Javie 9

Od Javy 9 nie można stosować podkreślenia (_) jako jedynego znaku definiującego identyfikator.

Treść klasy

Lewy nawias klamrowy (na końcu wiersza 4.), **{**, rozpoczyna **treść** deklaracji klasy. Deklaracja klasy musi kończyć się odpowiadającym mu **prawym nawiasem klamrowym** (wiersz 9.), **}**. Wiersze od 5. do 8. mają wcięcie.

2.4. Dobra praktyka programistyczna

Zastosuj jeden poziom wcięcia dla deklaracji każdej klasy dla kodu wewnątrz nawiasów klamrowych. W ten sposób podkreślisz strukturę deklaracji klasy i ułatwisz czytanie pliku. Wykorzystujemy trzy spacje jako formę wcięcia, choć wielu programistów preferuje dwie lub cztery spacje. Nieważne, co wybierzesz, ale swój wybór stosuj konsekwentnie.

2.5. Dobra praktyka programistyczna

IDE najczęściej automatycznie wykonują wcięcie kodu. Do stosowania wcięć służy też klawisz Tab. *Można skonfigurować IDE, aby dodawało preferowaną liczbę spacji po każdym naciśnięciu klawisza* Tab.

2.3. Typowy błąd programistyczny

Jeśli nawiasy nie są ułożone parami, zostanie zgłoszony błąd składniowy.

2.1. Wskazówka zapobiegająca błędom

Gdy użyjesz otwierającego nawiasu klamrowego, od razu wpisz też nawias zamykający, a dopiero później zacznij pisać treść wewnątrz nawiasów. Ta sztuczka pozwoli Ci uniknąć błędów związanych z niezamkniętymi nawiasami. Wiele IDE wykona ten krok za Ciebie.

Deklaracja metody

Wiersz 5.:

```
// Metoda main rozpoczyna wykonywanie aplikacji Javy
```

to komentarz wyjaśniający działanie wierszy od 6. do 8. Wiersz 6.:

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

to punkt początkowy każdej aplikacji Javy. **Nawiasy** po identyfikatorze `main` wskazują, że rozpoczyna się nowy blok, nazywany **metodą**. Deklaracja klasy Javy standardowo zawiera jedną lub więcej metod. W przypadku aplikacji Javy jedna z metod **musi** nosić nazwę `main` i musi być zdefiniowana w taki sam sposób jak w wierszu 6.

Metody wykonują zadania i mogą zwracać informacje, gdy je zakończą. Znaczenie słowa kluczowego `static` wyjaśnimy w punkcie 3.2.5. Słowo kluczowe **`void`** oznacza, że metoda **nie** zwróci żadnych informacji. W dalszej części pokażemy metodę, która zwraca informacje. Na razie po prostu stosuj przedstawiony wiersz i deklarację metody `main` w swoich programach. Fragment `String[] args` znajdujący się w nawiasach to wymagana część deklaracji `main` — zostanie opisany w rozdziale 7.

Lewy nawias klamrowy na końcu wiersza 6. rozpoczyna **treść deklaracji metody**. Kończy ją odpowiadający mu prawy nawias klamrowy (wiersz 8.). Wiersz 7. ma dodatkowe wcięcie.

2.6. Dobra praktyka programistyczna

Dokonaj dodatkowego wcięcia o jeden poziom względem poprzedniego wcięcia dla całej treści metody. W ten sposób podkreślisz strukturę metody i ułatwisz czytanie pliku.

Wyświetlanie informacji poleceniem System.out.println

Wiersz 7.:

```
System.out.println("Witamy programistę Javy!");
```

instruuje komputer, aby wykonał akcję, a dokładniej wyświetlił na ekranie tekst umieszczony między cudzysłowami. Cudzysłowy **nie** zostaną wyświetlone. Cudzysłowy i znaki między nimi tworzą tak zwany **ciąg znaków**, nazywany również **literałem tekstowym**. Białe spacje wewnątrz tekstu **nie** są przez kompilator ignorowane. Ciąg znaków **nie może** obejmować kilku wierszy kodu, ale w dalszej części książki pokażemy, jak obejść to ograniczenie.

Obiekt **`System.out`**, który jest predefiniowany, to tak zwany **standardowy obiekt wyjścia**. Umożliwia wyświetlanie informacji w **oknie poleceń**, w którym działa program. W systemie Windows okno poleceń nosi nazwę *Wiersz polecenia*, a w systemach Linux lub macOS najczęściej jest to **terminal** lub **powłoka**. Wielu programistów mówi po prostu o **wierszu poleceń**.

Metoda **`System.out.println`** wyświetla **wiersz** tekstu w wierszu poleceń. Gdy `System.out.println` zakończy swe zadanie, umieści kursor (miejsce rozpoczęcia wyświetlania następnych znaków) w pierwszej kolumnie nowego wiersza. Efekt jest taki sam jak w sytuacji, gdy naciśniesz klawisz *Enter* w edytorze tekstu — kursor przejdzie do pierwszego znaku nowego wiersza.

Cały wiersz 7., włączając w to `System.out.println`, argument `"Witamy programistę Javy!"` w nawiasach i **średnik (;)**, nosi nazwę **instrukcji** lub **polecenia**. Metody najczęściej zawierają wiele instrukcji wykonujących odpowiednie zadanie. Większość instrukcji kończy się znakiem średnika.

Komentarze jednowierszowe na końcach nawiasów zamykających

Jako pomoc dla początkujących programistów po nawiasach zamykających umieszczamy komentarze jednowierszowe informujące o zakończeniu deklaracji określonego elementu. Na przykład wiersz 8.:

```
} // Koniec metody main
```

wskazuje na koniec metody `main`, a wiersz 9.:

```
} // Koniec klasy Welcome1
```

wskazuje na koniec klasy `Welcome1`. Każdy z komentarzy wskazuje, że to nawias zamykający kończący określony fragment. Te dodatkowe komentarze będziemy stosowali tylko w kodzie z tego rozdziału.

2.2.1. Kompilacja aplikacji

Jesteśmy gotowi do skompilowania i wykonania programu. Zakładamy, że korzystasz z narzędzi wiersza poleceń JDK, a nie IDE. Poniższe instrukcje zakładają, że kod z przykładami znajduje się w folderze *c:\przyklady* w systemie Windows albo w folderze użytkownika *Documents/przyklady* w systemach Linux lub macOS.

Aby przygotować się do kompilacji programu, otwórz wiersz poleceń lub konsolę i przejdź do folderu z programem. Większość systemów operacyjnych oferuje polecenie `cd`, pozwalające przejść do wskazanego folderu. Oto przykład polecenia dla systemu Windows:

```
cd c:\przyklady\rozdzial02\02_01
```

który spowoduje przejście do folderu *02_01*. W systemach Linux i macOS polecenie ma postać:

```
cd ~/Documents/przyklady/rozdzial02/02_01
```

i powoduje przejście do folderu *02_01*. Aby skompilować program, wpisz:

```
javac Welcome1.java
```

Jeśli program nie zawiera żadnych błędów kompilacji, polecenie utworzy plik o nazwie *Welcome1.class* (nazywany też **plikiem klasy** `Welcome1`), zawierający niezależny od platformy kod bajtowy reprezentujący aplikację. Gdy użyjemy polecenia `java` w celu uruchomienia aplikacji na konkretnej platformie, JVM zamieni kod bajtowy na kod zrozumiały dla systemu operacyjnego i sprzętu.

2.4. Typowy błąd programistyczny

Błąd kompilacji typu `class Welcome1 is public, should be declared in a file named Welcome1.java` wskazuje, że nazwa pliku nie odpowiada nazwie klasy publicznej (oznaczonej `public`) zawartej w pliku lub podano złą nazwę klasy w trakcie kompilacji.

Czasem w trakcie nauki programowania warto celowo „popsuć" działający program, aby poznać różne błędy kompilacji. Komunikaty te nie zawsze informują o konkretnej przyczynie błędu. Po napotkaniu błędu komunikat to tylko ogólna sugestia. Spróbuj usunąć średnik lub nawias z kodu z rysunku 2.1 i skompiluj kod ponownie, aby zobaczyć błąd, który się pojawi.

2.2. Wskazówka zapobiegająca błędom

Gdy kompilator zgłosi błąd składniowy, może on występować w innym wierszu niż wskazany. Najpierw sprawdź wiersz, o którym mówi komunikat. Jeśli nie znajdziesz tam błędu, szukaj kilka wierszy wyżej.

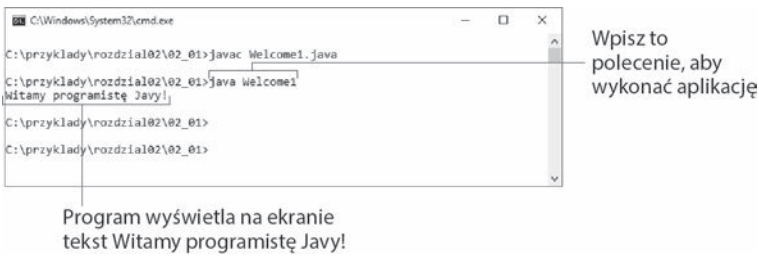
Każdy komunikat o błędzie kompilacji zawiera nazwę pliku i numer wiersza, w którym błąd wystąpił. Na przykład `Welcome1.java:6` wskazuje, że błąd wystąpił w wierszu 6. pliku *Welcome1.java*. Pozostała część komunikatu zawiera opis błędu składniowego.

2.2.2. Wykonywanie aplikacji

Po skompilowaniu programu wpisz następujące polecenie i naciśnij klawisz *Enter*:

```
java Welcome1
```

aby uruchomić maszynę wirtualną Javy i wczytać plik *Welcome1.class*. Polecenie **pomija** rozszerzenie pliku *.class*. W przeciwnym razie JVM **nie** wykonałoby programu. Następnie wiersz 7. programu wyświetla tekst `"Witamy programistę Javy!"`. Rysunek 2.2 przedstawia program wykonany w systemie Windows w programie *Wiersz polecenia*. (**Uwaga:** Większość środowisk stosuje czarne tło i biały tekst. Aby zwiększyć czytelność, dostosuj kolory).



Rysunek 2.2. Wykonywanie klasy Welcome1 w programie Wiersz polecenia

2.3. Wskazówka zapobiegająca błędom

Jeśli w trakcie uruchamiania programu Javy otrzymasz komunikat typu `Exception in thread "main" java.lang.NoClassDefFoundError: Welcome1`, zmienna środowiskowa `CLASSPATH` nie jest ustawiona prawidłowo. Ponownie przeczytaj instrukcję instalacji JDK znajdującą się na początku książki. Niektóre systemy wymagają ponownego uruchomienia, aby zmiany dotyczące `CLASSPATH` były widoczne.

2.3. Modyfikacja pierwszego programu

W tej części zmodyfikujemy przykład z rysunku 2.1, aby wyświetlić tekst w wierszu za pomocą wielu instrukcji, a także aby wyświetlić kilka wierszy tekstu za pomocą jednej instrukcji.

Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji

Tekst `Witamy programistę Javy!` można wyświetlić na wiele sposobów. Klasa `Welcome2` z rysunku 2.3 używa dwóch instrukcji (wiersz 7. i 8.), aby uzyskać efekt z rysunku 2.1. (**Uwaga:** Od tego momentu zmiany lub istotne fragmenty kodu będziemy w nowych listingach przedstawiać tak, jak wiersze 7. i 8.).

```
1 // Rysunek 2.3. Welcome2.java
2 // Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji
3
4 public class Welcome2 {
5    // Metoda main rozpoczyna wykonywanie aplikacji Javy
6    public static void main(String[] args) {
7       System.out.print("Witamy ");
8       System.out.println("programistę Javy!");
9    } // Koniec metody main
10 } // Koniec klasy Welcome2
```

```
Witamy programistę Javy!
```

Rysunek 2.3. Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji

Program jest bardzo podobny do tego z rysunku 2.1, więc omówimy jedynie różnice. Wiersz 2.:

```
// Wyświetlenie pojedynczego wiersza tekstu za pomocą kilku instrukcji
```

to komentarz jednowierszowy, który wyjaśnia działanie programu. Wiersz 4. rozpoczyna się od deklaracji klasy `Welcome2`. Wiersze 7. i 8. zawierające metodę `main`:

```
System.out.print("Witamy ");
System.out.println("programistę Javy!");
```

powodują wyświetlenie tylko **jednego** wiersza tekstu. Pierwsza instrukcja używa metody `print` z `System.out` do wyświetlania fragmentu tekstu. Każda instrukcja `print` lub `println` wznawia pisanie w miejscu, gdzie skończyła poprzednia. W odróżnieniu od `println` metoda `print` nie powoduje przeniesienia kursora do początku nowego wiersza — następny znak wyświetlany przez program znajduje się **dokładnie za** ostatnim znakiem wyświetlonym przez `print`. Oznacza to, że wiersz 8. umieści pierwszą literę (`"p"`) bezpośrednio po ostatnim znaku z wiersza 7. (**znak spacji** tuż przed cudzysłowem).

Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji

Pojedyncza instrukcja może wyświetlić kilka wierszy kodu, wykorzystując **znak przejścia do nowego wiersza** (`\n`), który powoduje przejście przez metody `print` i `println` z `System.out` do początku nowego wiersza. Podobnie jak znaki spacji i tabulacji, także znaki nowego wiersza stanowią białe spacje. Program z rysunku 2.4 wyświetla trzy wiersze tekstu, używając znaku nowego wiersza do przejścia do każdej nowej linii. Większość programu jest taka sama jak na rysunkach 2.1 i 2.3.

```
1 // Rysunek 2.4. Welcome3.java
2 // Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji
3
4 public class Welcome3 {
5    // Metoda main rozpoczyna wykonywanie aplikacji Javy
6    public static void main(String[] args) {
7       System.out.println("Witamy\nprogramistę\nJavy!");
8    } // Koniec metody main
9 } // Koniec klasy Welcome3
```

```
Witamy
programistę
Javy!
```

Rysunek 2.4. Wyświetlenie kilku wierszy tekstu za pomocą jednej instrukcji

Wiersz 7.:

```
System.out.println("Witamy\nprogramistę\nJavy!");
```

wyświetla w konsoli trzy wiersze tekstu. Standardowo znaki wewnątrz ciągu znaków są wyświetlane tak, jak je wpisano. Istnieją jednak pewne pary znaków (\ i `n` powtórzone w tekście dwukrotnie), które **nie** pojawiają się na ekranie. **Lewy ukośnik** (\) to znak ucieczki, który w metodach `print` i `println` z `System.out` ma znaczenie specjalne. Gdy lewy ukośnik pojawi się w tekście, Java łączy go ze znakiem następnym, aby utworzyć **sekwencję ucieczki** — w przypadku `\n` jest to znak nowego wiersza. Kiedy taki fragment pojawi się w wyświetlanym tekście, znak przejścia do nowego wiersza przeniesie kursor na początek nowego wiersza.

Rysunek 2.5 przedstawia listę kilku sekwencji ucieczki i opisuje sposób ich działania. Pełną listę wszystkich sekwencji ucieczki znajdziesz pod adresem:

http://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html - jls-3.10.6

Sekwencja ucieczki	Opis
`\n`	Znak nowego wiersza. Umieszcza kursor na początku **nowego** wiersza.
`\t`	Znak tabulacji poziomej. Przenosi kursor do następnego miejsca tabulacji.
`\r`	Powrót karetki. Umieszcza kursor na początku **aktualnego** wiersza — **nie** przechodzi do nowego wiersza. Dowolne znaki po nim spowodują **nadpisanie** znaków, które już znajdują się w danym wierszu.
`\\`	Lewy ukośnik. Używany do wyświetlenia na ekranie znaku lewego ukośnika.
`\"`	Cudzysłów. Używany do wyświetlenia na ekranie znaku cudzysłowu. Na przykład instrukcja: `System.out.println("\"w cudzysłowie\"");` spowoduje wyświetlenie tekstu `"w cudzysłowie"`.

Rysunek 2.5. Najczęściej używane sekwencje ucieczki

2.4. Wyświetlanie tekstu metodą printf

Metoda **`System.out.printf`** (`f` oznacza „formatowany”) wyświetla **formatowane** dane. Rysunek 2.6 przedstawia użycie jej do wyświetlenia dwóch osobnych tekstów: `"Witamy"` i `"programistę Javy!"`.

```
1 // Rysunek 2.6. Welcome4.java
2 // Wyświetlanie kilku wierszy metodą System.out.printf
3
4 public class Welcome4 {
5    // Metoda main rozpoczyna wykonywanie aplikacji Javy
6    public static void main(String[] args) {
7       System.out.printf("%s%n%s%n", "Witamy", "programistę Javy!");
8    } // Koniec metody main
9 } // Koniec klasy Welcome4
```

```
Witamy
programistę Javy!
```

Rysunek 2.6. Wyświetlanie kilku wierszy metodą System.out.printf

Wiersz 7.:

```
System.out.printf("%s%n%s%n", "Witamy", "programistę Javy!");
```

wywołuje metodę `System.out.printf` do wyświetlenia wyniku działania programu. Wywołanie metody korzysta z trzech argumentów. Gdy metoda wymaga kilku argumentów, umieszcza się je po kolei, **oddzielając przecinkami**. Wywołanie metody nazywa się czasem **zastosowaniem** metody.

2.7. Dobra praktyka programistyczna
Umieszczenie znaku spacji po każdym przecinku czyni listę argumentów czytelniejszą.

Pierwszym argumentem metody `printf` jest ciąg formatujący, który składa się ze **stałego tekstu** i **określników formatu**. Stały tekst jest wyświetlany przez `printf` w taki sam sposób jak przez metody `print` lub `println`. Każdy z określników formatu to **miejsce do zastąpienia** wartością, które wskazuje na **typ danych** wyjściowych. Określnik formatu może również zawierać dodatkowe informacje formatujące.

Określnik formatu rozpoczyna się znakiem procenta (`%`), po którym pojawia się znak reprezentujący **typ danych**. Przykładowo sekwencja **`%s`** oznacza miejsce do zastąpienia tekstem. W przedstawionym przykładzie ciąg formatujący wskazuje, że metoda `printf` powinna wyświetlić dwa teksty, a po każdym z nich przejść do nowego wiersza. Pierwszy określnik zostanie zastąpiony pierwszym argumentem po ciągu formatującym, a drugi następnym argumentem. W przykładzie tekst `"Witamy"` zastąpi pierwsze `%s`, a tekst `"programistę Javy!"` drugie `%s`. Każdy z tekstów pojawi się w swoim własnym wierszu.

Zwróć uwagę, że zamiast sekwencji ucieczki `\n` użyliśmy określnika formatu `%n`, który stosuje sekwencję przejścia do nowego wiersza **przenośną** między systemami. Nie można jej użyć jako argumentu w metodach `System.out.print` lub

`System.out.println`, ale przejście do nowego wiersza stosowane przez metodę `System.out.println` **po** wyświetleniu tekstu **jest** przenośne między systemami.

2.5. Inna aplikacja — dodawanie liczb całkowitych

Nasza następna aplikacja odczytuje dwie **liczby całkowite** (czyli liczby bez części ułamkowej, takie jak –22, 7, 0 lub 1024) wpisane przez użytkownika za pomocą klawiatury, dodaje je do siebie i wyświetla wynik. Program musi zapamiętać wartości podane przez użytkownika, aby móc je później zsumować. Programy pamiętają liczby i inne dane w pamięci komputera poprzez elementy nazywane **zmiennymi**. Program z rysunku 2.7 ilustruje to pojęcie. W przykładowym wyjściu użyliśmy kursywy z pogrubieniem do wskazania wartości wprowadzanych przez użytkownika (liczby ***45*** i ***72***). Podobnie jak w innych programach pierwszy i drugi wiersz zawiera odpowiednio nazwę pliku i opis sposobu działania.

```
1 // Rysunek 2.7. Addition.java
2 // Program dodający dwie wartości i wyświetlający wynik
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Addition {
6    // Metoda main rozpoczyna wykonywanie aplikacji Javy
7    public static void main(String[] args) {
8       // Tworzy obiekt Scanner, aby pobrać dane z konsoli
9       Scanner input = new Scanner(System.in);
10
11      System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
12      int number1 = input.nextInt(); // Odczytanie pierwszej liczby
13
14      System.out.print("Wpisz drugą liczbę: "); // Zapytanie
15      int number2 = input.nextInt(); // Odczytanie drugiej liczby
16
17      int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
18
19      System.out.printf("Suma wynosi %d%n", sum); // Wyświetlenie sumy
20   } // Koniec metody main
21 } // Koniec klasy Addition
```

```
Wpisz pierwszą liczbę: 45
Wpisz drugą liczbę: 72
Suma wynosi 117
```

Rysunek 2.7. Program dodający dwie wartości i wyświetlający wynik

2.5.1. Deklaracja import

Ogromną zaletą Javy jest bogaty zbiór predefiniowanych klas, z których można korzystać **wielokrotnie** i nie trzeba wymyślać koła na nowo. Klasy są pogrupowane w **pakiety — nazwane grupy powiązanych ze sobą klas** — a ich zbiór nazywa się **biblioteką klas Javy** lub **interfejsem programistycznym Javy** (**API Javy**). Wiersz 3.:

```
import java.util.Scanner; // Program używa klasy Scanner
```

to **deklaracja `import`**, która pozwala określić lokalizację klasy wykorzystywanej przez program. Wskazuje, że program używa predefiniowanej klasy `Scanner` (o której za chwilę) z pakietu o nazwie **`java.util`**. Kompilator upewni się, czy klasa jest stosowana prawidłowo.

2.5. Typowy błąd programistyczny

Wszystkie deklaracje `import` muszą pojawić się w pliku przed pierwszą deklaracją klasy. Umieszczenie deklaracji `import` w środku lub po deklaracji klasy jest błędem składniowym.

2.6. Typowy błąd programistyczny

Jeśli zapomnisz dodać deklarację `import`, spowoduje to błąd kompilacji z komunikatem typu `cannot find symbol`. Gdy tak się stanie, sprawdź, czy użyłeś deklaracji i czy wszystkie nazwy klas i pakietów są podane prawidłowo (włączając w to wielkość liter).

2.5.2. Deklaracja i utworzenie obiektu Scanner w celu pobrania danych z klawiatury

Zmienna to miejsce w pamięci komputera, w którym to wartość może być przechowywana w celu późniejszego użycia. Wszystkie zmienne Javy muszą być zadeklarowane za pomocą **nazwy** i **typu przed** ich pierwszym użyciem. **Nazwa** zmiennej umożliwia programowi dostęp do **wartości** zmiennej znajdującej się w pamięci. Nazwa zmiennej to dowolny poprawny identyfikator, czyli dowolny ciąg liter, cyfr, podkreśleń (_) lub znaków dolara ($), który **nie** rozpoczyna się od cyfry i **nie** zawiera znaków spacji. Typ zmiennej określa rodzaj informacji przechowywanej w pamięci. Podobnie jak inne instrukcje, także deklaracja zmiennej kończy się znakiem średnika (;).

Wiersz 9. w metodzie `main`:

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

to instrukcja deklaracji zmiennej, która wskazuje nazwę (`input`) i typ (`Scanner`) zmiennej do wykorzystania w programie. Klasa **`Scanner`** (z pakietu `java.util`) umożliwia programowi odczyt danych (np. liczb lub tekstu) w celu ich użycia. Dane mogą pochodzić z wielu różnych źródeł, w tym z klawiatury lub dysku twardego. Zanim użyjemy klasy `Scanner`, musimy utworzyć jej obiekt i wskazać **źródło** danych.

Znak `=` w wierszu 9. wskazuje, że zmienna `input` powinna zostać **zainicjalizowana** (przygotowana do użycia w programie) w momencie deklaracji wartością wyrażenia znajdującego się na prawo od znaku równości — `new Scanner(System.in)`. Wyrażenie używa słowa kluczowego **`new`** do utworzenia obiektu `Scanner`, który odczytuje znaki wpisywane na klawiaturze. **Standardowy obiekt wejścia, `System.in`**, umożliwia aplikacji odczyt bajtów danych wpisywanych przez użytkownika. Obiekt `Scanner` zamienia bajty na odpowiednie typy (np. `int`) w celu ich użycia przez program.

2.8. Dobra praktyka programistyczna
Wybór odpowiednich nazw zmiennych spowoduje, że program będzie się w zasadzie sam dokumentował (czyli zrozumienie zasady działania programu będzie możliwe przez samą analizę jego kodu bez stosowania dodatkowej dokumentacji zewnętrznej lub komentarzy).

2.9. Dobra praktyka programistyczna
Zgodnie z przyjętą konwencją w identyfikatorach nazw zmiennych stosuje się zapis wielbłądzi, ale z małą pierwszą literą, np. `liczbaPierwsza`.

2.5.3. Prośba o wprowadzenie danych

Wiersz 11.:

```
System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
```

używa metody `System.out.print` do wyświetlenia tekstu `"Wpisz pierwszą liczbę: "`. Ten tekst to **zapytanie**, ponieważ powoduje, że użytkownik będzie musiał podjąć pewną akcję. Używamy metody `print` zamiast metody `println`, aby użytkownik wprowadzał dane w tym samym wierszu co zapytanie. Przypomnij sobie z podrozdziału 2.2 informację, że identyfikatory zaczynające się od wielkiej litery zazwyczaj dotyczą nazw klas. Klasa `System` stanowi część pakietu **`java.lang`**.

2.1. Obserwacja z poziomu inżynierii oprogramowania
Pakiet `java.lang` jest domyślnie zaimportowany w każdym programie Javy. Klasy należące do tego pakietu to jedyne klasy, dla których nie trzeba stosować deklaracji `import`.

2.5.4. Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury

Deklaracja zmiennej z wiersza 12.:

```
int number1 = input.nextInt(); // Odczytanie pierwszej liczby
```

deklaruje zmienną `number1`, aby przechowywała daną typu **`int`**, czyli liczbę całkowitą w postaci `72`, `–1127` lub `0`. Zakresem dopuszczalnych wartości dla typu `int` są liczby od –2 147 483 648 do +2 147 483 647. Wartości `int` oczywiście nie zawierają znaków spacji, ale dla zwiększenia czytelności dopuszcza się stosowanie w liczbach znaku podkreślenia, więc `60_000_000` reprezentuje wartość `int` równą 60 000 000.

Innymi rodzajami danych są `float` i **`double`**, które przechowują liczby zmiennoprzecinkowe, a także typ **`char`**, który przechowuje konkretny znak. Liczby zmiennoprzecinkowe to liczby mogące zawierać ułamek, np. `3.4`, `0.0` lub `-11.19` (w kodzie zamiast znaku przecinka używa się znaku kropki). Zmienna typu **`char`** reprezentuje pojedynczy znak, np. dużą literę (`A`), cyfrę (`7`), znak specjalny (np. `*` lub `%`), a także sekwencję ucieczki (np. znak tabulacji `\t`). Typy `int`, `float`, `double` i `char` są **typami podstawowymi**. Nazwy tych typów są słowami kluczowymi i muszą być zapisywane małymi literami. Dodatek D podsumowuje charakterystykę wszystkich ośmiu typów podstawowych (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float` i `double`).

Znak = w wierszu 12. wskazuje, że zmienna number1 typu int powinna zostać w momencie deklaracji zainicjalizowana wynikiem input.nextInt(). Metoda next ↪Int() obiektu Scanner pobiera z klawiatury liczbę całkowitą. Program czeka, aż użytkownik wpisze liczbę i naciśnie klawisz *Enter*, aby potwierdzić wartość.

Nasz program zakłada, że użytkownik wprowadzi wartość poprawną. Jeśli nie, pojawi się błąd logiczny i program przestanie działać. Rozdział 11. zawiera informacje na temat zabezpieczania programów przed tego rodzaju błędami. Mówi się, że program przygotowany na błędy jest **odporny na porażki**.

2.5.5. Pobranie drugiej liczby całkowitej

Wiersz 14.:

```
System.out.print("Wpisz drugą liczbę: "); // Zapytanie
```

odpytuje użytkownika o drugą liczbę całkowita. Wiersz 15.:

```
int number2 = input.nextInt(); // Odczytanie drugiej liczby
```

deklaruje zmienną number2 typu int i inicjalizuje ją drugą wartością całkowitą pobraną od użytkownika.

2.5.6. Użycie zmiennych w obliczeniach

Wiersz 17.:

```
int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
```

deklaruje zmienną sum typu int i inicjalizuje ją wynikiem operacji number1 + number2. Gdy program napotka operację dodawania, wykonuje obliczenia na podstawie wartości znajdujących się w zmiennych number1 i number2.

W przedstawionym kodzie operator dodawania jest **operatorem binarnym**, ponieważ wykorzystuje **dwa operandy** — number1 i number2. Fragment dotyczący obliczenia nazywamy **wyrażeniem**. W zasadzie wyrażenie to dowolny fragment instrukcji posiadający **wartość**. Wartością wyrażenia number1 + number2 jest **suma** liczb. Podobnie wartością wyrażenia input.nextInt() (wiersze 12. i 15.) jest liczba całkowita wpisana przez użytkownika.

2.10. Dobra praktyka programistyczna

Aby zwiększyć czytelność kodu, umieść znak spacji przed operatorem binarnym i po nim.

2.5.7. Wyświetlenie wyniku obliczeń

Po wykonaniu obliczeń wiersz 19.:

```
System.out.printf("Suma wynosi %d%n", sum); // Wyświetlenie sumy
```

używa metody System.out.printf do wyświetlenia wartości zmiennej sum. Określnik formatu **%d** to **miejsce do zastąpienia** wartością typu int (w tym przypadku wartością z sum). Wszystkie pozostałe znaki tekstu formatującego to stała treść. Metoda printf wyświetli więc tekst "Suma wynosi ", po nim wartość sum (w miejscu wskazanym przez %d), a na końcu użyje znaku przejścia do nowego wiersza.

Obliczenia można również wykonać **wewnątrz** instrukcji wywołującej printf. Moglibyśmy połączyć wiersze 17. i 19. w jedną instrukcję:

```
System.out.printf("Suma wynosi %d%n", (number1 + number2));
```

Nawiasy, w których umieszczone jest wyrażenie `number1 + number2`, są opcjonalne — służą jedynie podkreśleniu, że wartość **całego** wyrażenia jest umieszczona w określniku formatu `%d`. Takie nawiasy nazywa się **redundantnymi**.

2.5.8. Dokumentacja API Javy

Dla każdej klasy z API Javy będziemy wskazywali pakiet, z którego pochodzi. W ten sposób będziesz w stanie szybko znaleźć opisy pakietów i klas w oficjalnej dokumentacji. Dokumentację w wersji WWW znajdziesz pod adresem:

http://docs.oracle.com/javase/8/docs/api/index.html

Wersję do pobrania znajdziesz w części *Additional Resources* na stronie:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

2.5.9. Deklaracja i inicjalizacja zmiennej jako osobne instrukcje

Każda zmienna musi mieć wartość, **zanim** będzie można jej użyć w obliczeniach (lub innym wyrażeniu). Instrukcja z wiersza 12. dokonywała zarówno deklaracji zmiennej `number1`, **jak i** jej inicjalizacji wartością wpisaną przez użytkownika.

Zdarza się, że deklaracja zmiennej następuje w jednej instrukcji, a w drugiej jest inicjalizowana. Na przykład wiersz 12. można by rozbić na dwie instrukcje:

```
int number1; // Deklaracja zmiennej number1 typu int
number1 = input.nextInt(); // Przypisanie zmiennej number1 danych pobranych
                           // od użytkownika
```

Pierwsza instrukcja deklaruje zmienną `number1`, ale jej **nie** inicjalizuje. Druga instrukcja używa **operatora przypisania** (`=`) do **przypisania** (czyli nadania) zmiennej `number1` wartości wprowadzonej przez użytkownika. Instrukcję można przeczytać jako rozkaz „**nadaj** `number1` wartość pochodzącą z `input.nextInt()`”. Wszystko, co znajduje się po **prawej** stronie operatora przypisania, jest zawsze wyliczane **przed** wykonaniem samego przypisania.

2.6. Zagadnienia dotyczące pamięci

Nazwy zmiennych takie jak `number1`, `number2` i `sum` tak naprawdę odpowiadają **miejscom** w pamięci komputera. Każda zmienna ma **nazwę**, **typ**, **rozmiar** (w bajtach) i **wartość**.

Gdy wykona się następujący fragment kodu z rysunku 2.7 (wiersz 12.):

```
int number1 = input.nextInt(); // Odczytanie pierwszej liczby
```

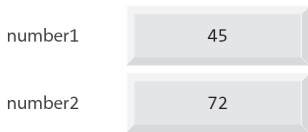
liczba wpisana przez użytkownika trafi do miejsca w pamięci wskazywanego przez nazwę `number1`. Przypuśćmy, że użytkownik wpisze `45`. Komputer umieści liczbę w położeniu `number1` (rysunek 2.8), zastępując nią poprzednią wartość w tym miejscu (jeśli istniała). Poprzednia wartość jest tracona, więc mówimy, że cały proces jest **destrukcyjny**.

Gdy wykona się instrukcja z wiersza 15.:

```
int number2 = input.nextInt(); // Odczytanie drugiej liczby
```

Rysunek 2.8. Miejsce w pamięci przedstawiające nazwę i wartość zmiennej number1

przypuśćmy, że jako wynik otrzymaliśmy 72. Komputer umieszcza liczbę całkowitą w miejscu wskazywanym przez zmienną number2. Pamięć wygląda obecnie tak, jak na rysunku 2.9.

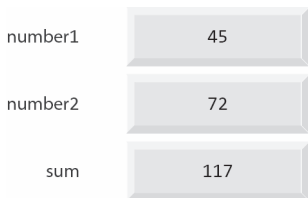


Rysunek 2.9. Miejsce w pamięci dla zmiennych number1 i number2

Po tym, jak program z rysunku 2.7 otrzymał obie wartości i umieścił je w zmiennych number1 i number2, dodaje je do siebie i sumę umieszcza w zmiennej sum. Instrukcja (wiersz 17.):

```
int sum = number1 + number2; // Dodanie liczb i zapamiętanie wyniku w sum
```

wykonuje dodawanie i zastępuje wynikiem dowolną wcześniejszą wartość znajdującą się w sum. Po tej operacji pamięć wygląda tak, jak na rysunku 2.10. Wartości zmiennych number1 i number2 wyglądają dokładnie tak samo jak wcześniej pomimo użycia ich do wyliczenia sum. Wartości te były używane, ale **nie** niszczone, w trakcie operacji dodawania. Gdy odczytujemy wartość z pamięci, jest to operacja **niedestrukcyjna**.



Rysunek 2.10. Miejsce w pamięci po zsumowaniu wartości ze zmiennych number1 i number2

2.7. Operacje arytmetyczne

Większość programów wykonuje operacje arytmetyczne. Podsumowanie dostępnych operatorów arytmetycznych zawiera rysunek 2.11. Zwróć uwagę na pewne symbole specjalne niestosowane w algebrze. **Znak gwiazdki** (*) oznacza mnożenie, a znak procenta (%) **resztę z dzielenia**, o którym więcej wkrótce. Operatory z rysunku 2.11 to operatory **binarne**, ponieważ wykorzystują **dwa** operandy. Na przykład wyrażenie f + 7 zawiera operator binarny + i dwa operandy: f i 7.

Dzielenie liczb całkowitych powoduje powstanie całkowitego wyniku. Na przykład wyrażenie 7 / 4 zwraca wynik 1, a 17 / 5 wynik 3. Część ułamkowa jest po prostu **ucinana (pomijana)** — nie dochodzi do **zaokrąglenia**. Java udostępnia operator reszty z dzielenia (%), który zwraca uciętą część. Wyrażenie x % y

Operacja	Operator	Wyrażenie algebraiczne	Wyrażenie w Javie
dodawanie	`+`	$f + 7$	`f + 7`
odejmowanie	`-`	$p - c$	`p - c`
mnożenie	`*`	bm	`b * m`
dzielenie	`/`	x / y lub $\frac{x}{y}$ albo $x \div y$	`x / y`
reszta z dzielenia	`%`	$r \bmod s$	`r % s`

Rysunek 2.11. Operatory arytmetyczne

zwróci resztę z dzielenia `x` przez `y`. Wynikiem operacji `7 % 4` będzie `3`, a wynikiem operacji `17 % 5` będzie `2`. Operator jest używany głównie dla liczb całkowitych, ale można go użyć również w połączeniu z innymi typami. W ćwiczeniach dotyczących tego rozdziału i następnych rozważymy kilka interesujących zastosowań operatora reszty z dzielenia, na przykład sprawdzenie, czy pewna liczba jest wielokrotnością innej.

Wyrażenia arytmetyczne zapisywane w linii prostej

Wyrażenia arytmetyczne w Javie trzeba zapisywać w linii prostej, czyli wersji przyjaznej dla komputerów. Oznacza to, że wyrażenie „`a` dzielone przez `b`" zapisuje się w postaci `a / b`, więc wszystkie stałe, zmienne i operatory pojawiają się w linii prostej. Kompilatory nie akceptują następującego zapisu algebraicznego:

$$\frac{a}{b}$$

Nawiasy do grupowania podwyrażeń

Nawiasy służą do grupowania wyrażeń w języku Java dokładnie w taki sam sposób jak w standardowych działaniach matematycznych. Aby pomnożyć `a` przez sumę `b + c`, napiszemy:

```
a * (b + c)
```

Jeśli wyrażenie zawiera zagnieżdżone nawiasy, jak tutaj:

```
((a + b) * c)
```

najpierw wyliczane są wartości **najbardziej wewnętrznych** nawiasów (w tym przypadku `a + b`).

Kolejność wykonywania działań

Java stosuje operatory arytmetyczne w ściśle określonej kolejności nazywanej **kolejnością wykonywania działań,** która w większości sytuacji jest taka sama jak w przypadku algebry.

1. Najpierw wykonuje się operacje mnożenia, dzielenia i reszty z dzielenia. Jeśli wyrażenie zawiera kilka tego rodzaju operacji, są one realizowane od lewej do prawej. Operacje mnożenia, dzielenia i reszty z dzielenia mają taki sam priorytet.
2. Następnie wykonuje się operacje dodawania i odejmowania. Jeśli wyrażenie zawiera kilka tego typu operacji, są one realizowane od lewej do prawej. Operacje dodawania i odejmowania mają taki sam priorytet.

Zasady te umożliwiają Javie stosowanie operatorów we właściwym **porządku**[1]. Gdy operatory są stosowane od lewej do prawej, mówimy o ich **łączności (asocjacyjności)**. Są jednak operatory, które łączą się od prawej do lewej. Rysunek 2.12 zawiera podsumowanie wszystkich reguł dla poznanych do tej pory operatorów. Kompletną listę operatorów zawiera dodatek A.

Operatory	Operacje	Kolejność wykonywania
*	mnożenie	Wyliczane w pierwszej kolejności. Jeśli istnieje kilka operatorów tego typu, są wyliczane od **lewej do prawej**.
/	dzielenie	
%	reszta z dzielenia	
+	dodawanie	Wyliczane w drugiej kolejności. Jeśli istnieje kilka operatorów tego typu, są wyliczane od **lewej do prawej**.
-	odejmowanie	
=	przypisanie	Wykonywane na końcu.

Rysunek 2.12. Kolejność wykonywania działań

Przykładowe wyrażenia algebraiczne i wyrażenia w języku Java

Przyjrzyjmy się kilku przykładom wyrażeń. Każde z nich przedstawione jest w wersji ze standardowym zapisem matematycznym i równoważnej wersji w języku Java. Oto przykład wyliczający średnią pięciu zmiennych:

Algebra: $m = \frac{a+b+c+d+e}{5}$

Java:
```
m = (a + b + c + d + e) / 5;
```

Nawiasy są niezbędne, ponieważ dzielenie ma wyższy priorytet niż dodawanie. Cały element `(a + b + c + d + e)` musi zostać podzielony przez 5. Gdybyśmy pominęli nawiasy, otrzymalibyśmy zapis `a + b + c + d + e / 5`, który byłby równoważny wersji

$$a+b+c+d+\frac{e}{5}$$

Oto przykład wyrażenia zapisywanego w jednym wierszu:

Algebra: $y = mx + b$

Java:
```
y = m * x + b;
```

Tutaj nawiasy nie są wymagane. Operacja mnożenia jest realizowana jako pierwsza, bo ma wyższy priorytet niż dodawanie. Przypisanie jest realizowane na końcu, bo ma niższy priorytet niż mnożenie lub dodawanie.

Poniższy przykład wykorzystuje operatory reszty z dzielenia (%), mnożenia, dzielenia, dodawania i odejmowania:

[1] Używamy prostych przykładów do przedstawienia kolejności wykonywania działań. W bardziej złożonych wyrażeniach mogą pojawić się subtelne problemy z kolejnością. Więcej informacji na ten temat znajdziesz w rozdziale 15. specyfikacji języka Java (*https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html*).

Algebra: $z = pr\%q + w / x - y$

```
Java: = p * r % q + w / x - y;
      6   1   2   4   3   5
```

Liczby w kółkach pod instrukcją wskazują **kolejność**, w jakiej Java zastosuje operatory. Operacje *, % i / są wykonywane jako pierwsze w kolejności **od lewej do prawej** (łączą się od lewej do prawej), ponieważ mają wyższy priorytet niż operacje + i –. Następnie są wykonywane operacje + i –. One też są realizowane **od lewej do prawej**. Operator przypisania (=) wchodzi do gry jako ostatni.

Wyliczanie wielomianu stopnia drugiego

Aby lepiej zrozumieć kolejność wykonywania działań, przeanalizujmy obliczanie wartości wyrażenia, które jest wielomianem stopnia drugiego w postaci $ax^2 + bx + c$:

```
y = a * x * x + b * x + c;
6   1   2   4   3   5
```

Operacje mnożenia są wykonywane w pierwszej kolejności i od lewej do prawej, bo mają wyższy priorytet niż dodawanie. (Ponieważ Java nie ma operatora arytmetycznego dla podnoszenia do potęgi, x^2 zapisaliśmy jako `x * x`. Alternatywę dla tej operacji zawiera podrozdział 5.4). Operacje dodawania są wykonywane **od lewej do prawej**. Przypuśćmy, że zmienne `a`, `b`, `c` i `x` zainicjalizowano (nadano im wartości) w następujący sposób: `a = 2`, `b = 3`, `c = 7` i `x = 5`. Rysunek 2.13 przedstawia kolejność, w której będą stosowane operatory.

```
Krok 1.  y = 2 * 5 * 5 + 3 * 5 + 7;   (mnożenie najbardziej z lewej)
           2 * 5 to 10
Krok 2.  y = 10 * 5 + 3 * 5 + 7;      (mnożenie najbardziej z lewej)
           10 * 5 to 50
Krok 3.  y = 50 + 3 * 5 + 7;          (mnożenie przed dodawaniem)
             3 * 5 to 15
Krok 4.  y = 50 + 15 + 7;             (dodawanie najbardziej z lewej)
           50 + 15 to 15
Krok 5.  y = 65 + 7;                  (ostatnie dodawanie)
           65 + 7 to 15
Krok 6.  y = 72                       (ostatnia operacja, czyli umieszczenie 72 w y)
```

Rysunek 2.13. Kolejność wyliczania wartości w wielomianie stopnia drugiego

Można zastosować nieobowiązkowe nawiasy, aby uczynić wyrażenie bardziej czytelnym. Analizowane wyrażenie można zapisać następująco:

```
y = (a * x * x) + (b * x) + c;
```

2.8. Podejmowanie decyzji — operatory równości i relacji

Warunek to wyrażenie, które może być **prawdziwe** lub **fałszywe**. W tym podrozdziale omówimy **instrukcję warunkową `if`**, która umożliwia programowi podejmowanie **decyzji** na podstawie zawartości zmiennej. Na przykład warunek „suma punktów jest większa lub równa 60" określa, czy student zdał test. Jeśli warunek z instrukcji `if` jest **prawdziwy**, wykona się treść zawarta w instrukcji. Jeśli jest **fałszywy**, zostanie pominięta.

Warunki w instrukcjach `if` mogą powstawać z użyciem **operatorów równości** (`==` i `!=`) lub **operatorów relacji** (`>`, `<`, `>=` i `<=`) podsumowanych na rysunku 2.14. Oba operatory równości mają taki sam priorytet, który jest **niższy** od priorytetu operatorów relacji. Operatory równości łączą się od **lewej do prawej**. Operatory relacji mają taki sam priorytet i również łączą się od **lewej do prawej**.

Operator matematyczny	Operator równości lub relacji w Javie	Przykładowy warunek w Javie	Znaczenie warunku w Javie
Operatory równości			
$=$	`==`	`x == y`	x jest równe y
$\neq$	`!=`	`x != y`	x jest różne od y
Operatory relacji			
$>$	`>`	`x > y`	x jest większe od y
$<$	`<`	`x < y`	x jest mniejsze od y
$\geq$	`>=`	`x >= y`	x jest większe lub równe y
$\leq$	`<=`	`x <= y`	x jest mniejsze lub równe y

Rysunek 2.14. Operatory równości i relacji

Na rysunku 2.15 zostało użytych sześć instrukcji `if` do porównywania dwóch liczb całkowitych wpisanych przez użytkownika. Jeśli warunek w którejkolwiek instrukcji będzie **prawdziwy**, instrukcja powiązana z instrukcją `if` wykona się; w przeciwnym razie zostanie pominięta. Używamy klasy `Scanner` do pobrania liczb całkowitych od użytkownika i umieszczenia ich w zmiennych `number1` i `number2`. Program **porównuje** liczby i wyświetla informacje, jeżeli wynik porównywania jest prawdziwy. Przedstawiamy również trzy wyniki dla różnych wartości wpisanych przez użytkownika.

```
1 // Rysunek 2.15. Comparison.java
2 // Porównywanie liczb za pomocą instrukcji if, operatorów równości i operatorów relacji
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Comparison {
6     // Metoda main rozpoczyna wykonywanie aplikacji Java
```

Rysunek 2.15. Porównywanie liczb za pomocą instrukcji if, operatorów równości i operatorów relacji

```
   public static void main(String[] args) {
      // Tworzy obiekt Scanner, aby pobrać dane z konsoli
      Scanner input = new Scanner(System.in);

      System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
      int number1 = input.nextInt(); // Odczytanie pierwszej liczby

      System.out.print("Wpisz drugą liczbę: "); // Zapytanie
      int number2 = input.nextInt(); // Odczytanie pierwszej liczby

      if (number1 == number2) {
         System.out.printf("%d == %d%n", number1, number2);
      }

      if (number1 != number2) {
         System.out.printf("%d != %d%n", number1, number2);
      }

      if (number1 < number2) {
         System.out.printf("%d < %d%n", number1, number2);
      }

      if (number1 > number2) {
         System.out.printf("%d > %d%n", number1, number2);
      }

      if (number1 <= number2) {
         System.out.printf("%d <= %d%n", number1, number2);
      }

      if (number1 >= number2) {
         System.out.printf("%d >= %d%n", number1, number2);
      }
   } // Koniec metody main
} // Koniec klasy Comparison
```

```
Wpisz pierwszą liczbę: 777
Wpisz drugą liczbę: 777
777 == 777
777 <= 777
777 >= 777
```

```
Wpisz pierwszą liczbę: 1000
Wpisz drugą liczbę: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Wpisz pierwszą liczbę: 2000
Wpisz drugą liczbę: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Rysunek 2.15. Porównywanie liczb za pomocą instrukcji if, operatorów równości i operatorów relacji — *cigg dalszy*

Metoda `main` (wiersze od 7. do 40.) klasy `Comparison` rozpoczyna wykonywanie programu. Wiersz 9.:

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

deklaruje zmienną `input` typu `Scanner` i przypisuje jej obiekt `Scanner` powiązany ze standardowym wejściem (np. klawiaturą).

Wiersze 11. i 12.:

```
System.out.print("Wpisz pierwszą liczbę: "); // Zapytanie
int number1 = input.nextInt(); // Odczytanie pierwszej liczby
```

pytają użytkownika o pierwszą liczbę całkowitą i zapisują ją w zmiennej `number1` typu `int`.

Wiersze 14. i 15.:

```
System.out.print("Wpisz drugą liczbę: "); // Zapytanie.
int number2 = input.nextInt(); // Odczytanie drugiej liczby.
```

pytają użytkownika o drugą liczbę całkowitą i zapisują ją w zmiennej `number2` typu `int`.

Wiersze od 17. do 19.:

```
if (number1 == number2) {
   System.out.printf("%d == %d%n", number1, number2);
}
```

porównują zawartość zmiennych `number1` i `number2` pod kątem równości. Jeśli wartości są sobie równe, instrukcja z wiersza 18. wyświetla tekst wskazujący, że wartości są równe. Instrukcje `if` z wierszy 21., 25., 29., 33. i 37. porównują `number1` i `number2`, wykorzystując operatory `!=`, `<`, `>`, `<=` i `>=`. Jeśli warunki są prawdziwe w jednej z tych instrukcji, wówczas wyświetli się tekst umieszczony poniżej warunku.

Każda instrukcja `if` z rysunku 2.15 zawiera jako swoją treść tylko jedno wcięte polecenie. Zauważ, że treść znajduje się w nawiasach klamrowych ({ i }). Tworzy to tak zwaną **instrukcję złożoną** lub **blok instrukcji**.

2.11. Dobra praktyka programistyczna

Zastosuj wcięcie dla instrukcji znajdujących się wewnątrz instrukcji `if`, aby poprawić czytelność kodu. IDE najczęściej wykonują to zadanie automatycznie po określeniu wielkości wcięcia.

2.4. Wskazówka zapobiegająca błędom

Jeśli treścią wykonywaną dla warunku jest tylko jedno polecenie, nie trzeba używać nawiasów klamrowych. Są one niezbędne tylko w sytuacji, gdy poleceń jest więcej. Wkrótce się przekonasz, że zapominanie o stosowaniu nawiasów klamrowych w przypadku wielu instrukcji wewnątrz treści prowadzi do subtelnych błędów. Aby ich uniknąć, zawsze stosuj nawiasy klamrowe dla instrukcji `if`.

Białe spacje

Zwróć uwagę na zastosowanie białych spacji na rysunku 2.15. Przypomnijmy, że kompilator najczęściej ignoruje białe spacje. Jeśli instrukcje zajmują kilka wierszy, warto rozdzielić je w sposób logiczny, co poprawia czytelność, ale nie wpływa

2.7. Typowy błąd programistyczny

Umieszczanie średnika tuż po nawiasie zamykającym warunek w instrukcji `if` *jest często błędem logicznym (ale nie błędem składniowym). Średnik powoduje, że treść instrukcji* `if` *jest tak naprawdę pusta, więc spełnienie warunku nie spowoduje żadnej akcji. Co gorsza, oryginalna treść warunku wykona się* ***zawsze*** *(niezależnie od warunku), co spowoduje zapewne błędne działanie programu.*

2.5. Wskazówka zapobiegająca błędom

Długie instrukcje mogą swym zasięgiem obejmować kilka wierszy. Jeśli taki podział jest konieczny, użyj naturalnych miejsc złamania wiersza, na przykład po przecinku w przypadku listy elementów lub po operatorze w przypadku długiego wyrażenia. Jeżeli instrukcja jest podzielona na kilka wierszy, stosuj wcięcia aż do jej końca.

na działanie aplikacji. W idealnej sytuacji liczba instrukcji w metodzie powinna być stosunkowo niewielka, ale nie zawsze jest to możliwe.

Opisane dotychczas operatory

Rysunek 2.16 przedstawia omówione dotychczas operatory od najwyższego do najniższego priorytetu. Wszystkie poza operatorem przypisania (=) dotyczą łączenia **od lewej do prawej**. Operator przypisania wykonuje się **od prawej do lewej**. To, co trafia do zmiennej wskazanej po lewej stronie jako efekt działania wyrażenia znajdującego się po prawej stronie operatora przypisania, jest też wynikiem całego operatora — na przykład wartością wyrażenia `x = 7` jest `7`. Wyrażenie takie jak `x = y = 0` wylicza się tak, jakby było zapisane jako `x = (y = 0)`, czyli najpierw następuje przypisanie wartości 0 do zmiennej `y`, a dopiero potem przypisanie wyniku tego przypisania (0) do zmiennej `x`.

Operatory	Asocjacyjność	Typ
`* / %`	od lewej do prawej	mnożące
`+ -`	od lewej do prawej	dodające
`< <= > >=`	od lewej do prawej	relacyjne
`== !=`	od lewej do prawej	równości
`=`	od prawej do lewej	przypisania

Rysunek 2.16. Priorytet i kierunek działania przedstawionych dotychczas operatorów

2.12. Dobra praktyka programistyczna

Pisząc wyrażenie zawierające wiele operatorów, skorzystaj z tabeli kolejności operatorów dostępnej w dodatku A. Sprawdź, czy operacje w wyrażeniu są wykonywane w kolejności, której oczekujesz. Jeśli w przypadku złożonego wyrażenia nie jesteś pewien kolejności, użyj z nawiasów, aby wymusić pożądaną kolejność wykonywania operacji.

2.9. Podsumowanie

W tym rozdziale poznałeś naprawdę dużo rozmaitych funkcjonalności Javy, w tym wyświetlanie danych na ekranie, wprowadzanie danych z klawiatury, wykonywanie obliczeń i podejmowanie decyzji. Przedstawione tu aplikacje wprowadzają wiele podstawowych pojęć programistycznych. Przekonasz się podczas lektury następnego rozdziału, że aplikacje Javy zawierają w metodzie `main` jedynie kilka wierszy kodu, ponieważ większość prac aplikacja wykonuje przez tworzenie obiektów i wywoływanie metod. Z rozdziału 3. dowiesz się, jak tworzyć własne klasy i używać utworzonych na ich podstawie obiektów.

Streszczenie

Podrozdział 2.2. Twój pierwszy program — wyświetlenie wiersza tekstu

- Aplikację Javy uruchamia się, wykonując polecenie `java` w celu uruchomienia JVM.
- Komentarze dokumentują program i zwiększają jego czytelność. Kompilator je ignoruje.
- Komentarz jednowierszowy zaczyna się znakami // i kończy wraz z końcem wiersza, w którym się znajduje.
- Komentarz tradycyjny może obejmować swym zasięgiem wiele wierszy. Zaczyna się znakami /* i kończy znakami */.
- Komentarz Javadoc, zaczynający się znakami /** i kończący znakami */, pozwala osadzać dokumentację bezpośrednio w kodzie. Program `javadoc` umożliwia generowanie na podstawie tych komentarzy dokumentacji projektowej.
- Błąd składniowy występuje, gdy kompilator natknie się na kod, który łamie reguły języka Java. Przypomina to błąd gramatyczny w języku naturalnym.
- Puste wiersze, znaki spacji i znaki tabulacji to tak zwane białe spacje. Dzięki nim program jest czytelniejszy, ale nie wpływa to na jego działanie (kompilator takie znaki ignoruje).
- Słowa kluczowe to nazwy zarezerwowane przez Javę. Zawsze są zapisywane małymi literami.
- Słowo kluczowe `class` rozpoczyna deklarację klasy.
- Zgodnie z przyjętą konwencją wszystkie nazwy klas Javy rozpoczynają się od wielkiej litery, a także zawierają wielką literę na początku każdego następnego wyrazu (np. `NazwaPewnejKlasy`).
- Nazwa klasy Javy to identyfikator — ciąg znaków składający się z liter, cyfr, znaku podkreślenia (_) i znaku dolara ($), który jednak nie zaczyna się od cyfry i nie zawiera spacji.
- Deklaracja klasy publicznej (`public`) musi być zapisana w pliku o takiej samej nazwie mającym rozszerzenie *.java*.
- Java to język, w którym istotna jest wielkość liter, więc „widzi” różnicę między małymi i dużymi literami.
- Treść deklaracji każdej klasy znajduje się w nawiasach klamrowych ({ i }).

- Metoda `main` to punkt początkowy każdej aplikacji. Musi się zaczynać w następujący sposób:

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

W przeciwnym razie JVM nie uruchomi aplikacji.

- Metody wykonują zadania i zwracają informacje, gdy je zakończą. Słowo kluczowe `void` wskazuje, że metoda wykonuje zadanie, ale nie zwraca informacji.
- Instrukcje powodują wykonywanie pewnych akcji przez komputer.
- Ciąg znaków umieszcza się w cudzysłowie i nazywa czasem literałem znakowym.
- Standardowy obiekt wyjścia (`System.out`) wyświetla znaki w oknie terminala.
- Metoda `System.out.println` wyświetla w terminalu tekst przekazany jako argument, a następnie przenosi kursor do początku nowego wiersza.

Punkt 2.2.1. Kompilacja aplikacji

- Kompilację programu przeprowadza się poleceniem `javac`. Jeśli program nie zawiera żadnych błędów składniowych, powstanie plik klasy zawierający kod bajtowy reprezentujący aplikację. Kod bajtowy jest interpretowany przez JVM w trakcie wykonywania programu.

Punkt 2.2.2. Wykonywanie aplikacji

- Aby uruchomić aplikację, użyj polecenia `java` i podaj po nim nazwę klasy zawierającej metodę `main`.

Podrozdział 2.3. Modyfikacja pierwszego programu

- Metoda `System.out.print` wyświetla przekazany argument i umieszcza kursor od razu za ostatnim wyświetlonym znakiem.
- Lewy ukośnik (\) to znak ucieczki. Java łączy go z następnym znakiem w sekwencję ucieczki — np. kombinacja `\n` reprezentuje znak przejścia do nowego wiersza.

Podrozdział 2.4. Wyświetlanie tekstu metodą printf

- Metoda `System.out.printf` (`f` oznacza „formatowane”) wyświetla dane w postaci sformatowanej.
- Pierwszym argumentem metody `printf` jest ciąg formatujący, który zawiera stały tekst lub określniki formatu. Każdy określnik formatu wskazuje typ danych wyjściowych i jest elementem tymczasowym informującym o konkretnym miejscu wstawienia danych.
- Określnik formatu rozpoczyna się znakiem procentu (`%`), po którym następuje znak definiujący typ danych, np. określnik formatu `%s` oznacza tekst.
- Określnik formatu `%n` to przenośne przejście do nowego wiersza. Nie można go użyć jako argumentu w metodach `System.out.print` lub `System.out.println`, ale metoda `System.out.println` na końcu tekstu również wstawia przejście do nowego wiersza w sposób przenośny między systemami.

Punkt 2.5.1. Deklaracja import

- Deklaracja import pomaga kompilatorowi odnaleźć klasę używaną przez program.
- Bogaty zestaw predefiniowanych klas Javy jest pogrupowany w pakiety — nazwane grupy klas. Najczęściej mówi się o bibliotece klas Javy lub API Javy.

Punkt 2.5.2. Deklaracja i utworzenie obiektu Scanner w celu pobrania danych z klawiatury

- Zmienna to miejsce w pamięci komputera, w którym znajduje się wartość używana później przez program. Wszystkie zmienne muszą być deklarowane przed pierwszym użyciem za pomocą nazwy i typu.
- Nazwa zmiennej umożliwia programowi odwołanie się do konkretnego miejsca w pamięci.
- Klasa `Scanner` (z pakietu `java.util`) umożliwia programowi odczyt danych z zewnątrz. Tworzony obiekt `Scanner` musi otrzymać jako argument źródło danych.
- Zmienne należy inicjalizować przed pierwszym użyciem przez program.
- Wyrażenie `new Scanner(System.in)` tworzy obiekt `Scanner`, który odczytuje dane ze standardowego obiektu wejścia (`System.in`).

Punkt 2.5.3. Prośba o wprowadzenie danych

- Zapytanie prosi użytkownika o podjęcie konkretnej akcji.

Punkt 2.5.4. Deklaracja zmiennej dla liczby całkowitej i pobranie wartości z klawiatury

- Typ danych `int` służy do deklarowania zmiennych mogących przechowywać wartości całkowite. Typ `int` dopuszcza wartości w zakresie od –2 147 483 648 do +2 147 483 647.
- Wartości typu `int` nie mogą w kodzie źródłowym zawierać spacji, ale ze względu na poprawę czytelności dopuszcza się stosowanie znaków podkreślenia (np. `60_000_000`).
- Typy `float` i `double` dotyczą obsługi liczb zmiennoprzecinkowych, takich jak `-11.19` lub `3.4`.
- Zmienne typu `char` reprezentują pojedyncze znaki, np. dużą literę (`A`), cyfrę (`7`), znak specjalny (np. `*` lub `%`), a także sekwencję ucieczki (np. znak tabulacji `\t`).
- Typy takie jak `int`, `float`, `double` i `char` to typy podstawowe. Typy podstawowe są słowami kluczowymi, więc zapisuje się je małymi literami.
- Metoda `nextInt` klasy `Scanner` pobiera od użytkownika liczbę całkowitą.

Punkt 2.5.6. Użycie zmiennych w obliczeniach

- Fragmenty instrukcji mające wartości nazywamy wyrażeniami.

Punkt 2.5.7. Wyświetlenie wyniku obliczeń

- Określnik formatu `%d` stanowi miejsce dla wartości typu `int`.

Punkt 2.5.9. Deklaracja i inicjalizacja zmiennej jako osobne instrukcje

- Zmienna musi mieć przypisaną wartość, aby mogła być używana przez program.
- Operator przypisania (`=`) umożliwia nadanie wartości zmiennej.

Podrozdział 2.6. Zagadnienia dotyczące pamięci

- Nazwy zmiennych odpowiadają pewnym obszarom w pamięci komputera. Każda zmienna ma nazwę, typ, rozmiar i wartość.
- Wartość wstawiana do zmiennej zastępuje wartość, która znajdowała się w danym miejscu pamięci wcześniej. Innymi słowy, stara wartość jest tracona.

Podrozdział 2.7. Operacje arytmetyczne

- Operatorami arytmetycznymi są: + (dodawanie), - (odejmowanie), * (mnożenie), / (dzielenie) i % (reszta z dzielenia).
- Dzielenie liczb całkowitych zwraca liczbę całkowitą bez części ułamkowej.
- Operator reszty z dzielenia (%) zwraca resztę pozostałą po dzieleniu liczb całkowitych.
- Wyrażenia arytmetyczne w kodzie trzeba pisać w wersji liniowej.
- Jeśli wyrażenie zawiera zagnieżdżone nawiasy, w pierwszej kolejności wyliczane są wartości z najbardziej wewnętrznych nawiasów.
- Java stosuje operatory arytmetyczne w wyrażeniach w ściśle określonej kolejności wskazywanej przez kolejność wykonywania działań.
- Gdy mówimy, że operatory są stosowane od lewej do prawej, mamy na myśli ich łączenie (asocjacyjność). Niektóre operatory są stosowane od prawej do lewej.
- Dodatkowe (zbędne) nawiasy czynią wyrażenia czytelniejszymi.

Podrozdział 2.8. Podejmowanie decyzji — operatory równości i relacji

- Instrukcja `if` podejmuje decyzję na podstawie spełnienia (prawda lub fałsz) wskazanego warunku.
- Warunki w instrukcji `if` wykorzystują operatory równości (`==` i `!=`) oraz operatory relacji (`>`, `<`, `<=` i `>=`).
- Instrukcja `if` rozpoczyna się od słowa kluczowego `if`, następnie jest umieszczony w nawiasach warunek, a potem treść w postaci dodatkowych instrukcji. Dodatkowe instrukcje, jeśli jest ich więcej niż jedna, muszą znajdować się w nawiasach klamrowych.

Ćwiczenia do samooceny

1.1. Wypełnij puste miejsca w każdym z następujących zdań:

a) _____ i _____ zaczynają i kończą treść każdej metody.
b) Instrukcja _____ służy do podejmowania decyzji.
c) Znaki _____ rozpoczynają komentarz jednowierszowy.
d) _____, _____ i _____ są nazywane białymi spacjami.
e) _____ są zarezerwowane przez język Java.
f) Aplikacje Javy rozpoczynają działanie od metody _____.
g) Metody _____, _____ i _____ wyświetlają informacje w oknie terminala.

1.2. Określ, które z poniższych zdań są **prawdziwe**, a które **fałszywe**. Jeśli są fałszywe, wyjaśnij dlaczego.

a) Komentarze powodują, że w trakcie wykonywania programu komputer wyświetla tekst umieszczony po znakach //.
b) Wszystkie zmienne muszą mieć wskazany typ w momencie ich deklarowania.
c) Java traktuje zmienne `liczba` i `LIczBa` jako identyczne.
d) Operator reszty z dzielenia (%) może być stosowany tylko dla operandów będących liczbami całkowitymi.
e) Operatory arytmetyczne *, /, %, + i - mają taki sam priorytet.
f) Identyfikator _ (podkreślenie) użyty samodzielnie jest poprawny w Javie 9.

1.3. Napisz instrukcje realizujące każde z wymienionych niżej zadań:

a) Zadeklaruj zmienne `c`, `thisIsAVariable`, `q76354` i `number` o typie `int` i zainicjalizuj każdą z nich wartością 0.
b) Poproś użytkownika o wpisanie liczby całkowitej.

c) Pobierz liczbę całkowitą i umieść ją w zmiennej `value` typu `int`. Załóż, że istnieje obiekt `Scanner` o nazwie `input` obsługujący odczyt wartości z klawiatury.

d) Wyświetl w oknie konsoli w jednym wierszu tekst `"To jest program Javy"`. Użyj metody `System.out.println`.

e) Wyświetl tekst `"To jest program Javy"` w dwóch wierszach w konsoli. Pierwszy wiersz powinien kończyć się wyrazem `program`. Użyj metody `System.out.printf` i dwóch określników formatu `%s`.

f) Jeśli zmienna `number` nie jest równa `7`, wyświetl tekst `"Zmienna number nie jest równa 7"`.

1.4. Znajdź i popraw błędy w każdym z następujących fragmentów kodu:

a)
```
if (c < 7); {
   System.out.println("c jest mniejsze od 7");
}
```

b)
```
if (c => 7) {
   System.out.println("c jest większe lub równe 7");
}
```

1.5. Napisz deklaracje, instrukcje lub komentarze, które realizują wszystkie wskazane zadania:

a) Wskaż, że program będzie wyliczał iloczyn trzech liczb całkowitych.

b) Utwórz obiekt `Scanner` w zmiennej `input`, który odczytuje wartości ze standardowego wejścia.

c) Poproś użytkownika o wpisanie pierwszej liczby całkowitej.

d) Odczytaj pierwszą liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `x` typu `int`.

e) Poproś użytkownika o wpisanie drugiej liczby całkowitej.

f) Odczytaj drugą liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `y` typu `int`.

g) Poproś użytkownika o wpisanie trzeciej liczby całkowitej.

h) Odczytaj trzecią liczbę całkowitą podaną przez użytkownika i zapamiętaj ją w zmiennej `z` typu `int`.

i) Oblicz iloczyn trzech liczb całkowitych znajdujących się w zmiennych `x`, `y` i `z`, a następnie umieść wynik w zmiennej `result` typu `int`.

j) Użyj metody `System.out.printf` do wyświetlenia tekstu `"Iloczyn wynosi"`, po którym pojawi się zawartość zmiennej `result`.

1.6. Wykorzystując polecenia z ćwiczenia 2.5, napisz kompletny program, który wylicza i wyświetla iloczyn trzech liczb całkowitych.

Odpowiedzi do samooceny

2.1. a) lewy nawias ({), prawy nawias (}); b) `if`; c) //; d) znaki spacji, nowych wierszy i tabulacji; e) słowa kluczowe; f) `main`; g) `System.out.print`, `System.out.println` i `System.out.printf`.

2.2. Odpowiedzi:

a) Fałsz. Komentarze nie powodują podejmowania przez program jakichkolwiek akcji. Dotyczą dokumentowania kodu i poprawy jego czytelności.

b) Prawda.

c) Fałsz. Java jest językiem, w którym ważna jest wielkość liter, więc zmienne są różne.

d) Fałsz. Operator reszty może być stosowany także dla operandów niebędących liczbami całkowitymi.

e) Fałsz. Operatory *, / i % mają wyższy priorytet niż operatory + i -.

f) Fałsz. Od Javy 9 _ (podkreślenie) nie jest już poprawnym identyfikatorem.

2.3. Odpowiedzi:

a)
```
int c = 0;
int thisIsAVariable = 0;
int q76354 = 0;
int number = 0;
```

b)
```
System.out.print("Wpisz liczbę całkowitą: ");
```

c)
```
int value = input.nextInt();
```

d)
```
System.out.println("To jest program Javy ");
```

e)
```
System.out.printf("%s%n%s%n", "To jest program", "Javy");
```

f)
```
if (number != 7) {
   System.out.println("Zmienna number nie jest równa 7");
}
```

2.4. Odpowiedzi:

a) Błąd polega na użyciu średnika tuż za nawiasem kończącym warunek (`c < 7`) w `if`. W efekcie kod poniżej wykona się niezależnie od tego, czy warunek okazuje się prawdziwy.
Sposób poprawy: usuń średnik po nawiasie.

b) Błąd polega na użyciu niepoprawnego operatora relacji `=>`.
Sposób poprawy: zamień `=>` na `>=`.

2.5. Odpowiedzi:

a)
```
// Oblicz iloczyn trzech liczb całkowitych.
```

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

c)
```
System.out.print("Wpisz pierwszą liczbę całkowitą: ");
```

d)
```
int x = input.nextInt();
```

e)
```
System.out.print("Wpisz drugą liczbę całkowitą: ");
```

f)
```
int y = input.nextInt();
```

g)
```
System.out.print("Wpisz trzecią liczbę całkowitą: ");
```

h)
```
int z = input.nextInt();
```

i)
```
int result = x * y * z;
```

j)
```
System.out.printf("Iloczyn wynosi %d%n", result);
```

2.6. Odpowiedź:

```
1 // Ćwiczenie 2.6. Product.java
2 // Oblicz iloczyn trzech liczb całkowitych
3 import java.util.Scanner; // Program używa klasy Scanner
4
5 public class Product {
6    public static void main(String[] args) {
7       // Utwórz obiekt Scanner, aby pobrać dane wejściowe z klawiatury
8       Scanner input = new Scanner(System.in);
9
10      System.out.print("Wpisz pierwszą liczbę całkowitą: "); // Zapytanie
11      int x = input.nextInt(); // Odczyt pierwszej liczby
12
13      System.out.print("Wpisz drugą liczbę całkowitą: "); // Zapytanie
14      int y = input.nextInt(); // Odczyt drugiej liczby
15
16      System.out.print("Wpisz trzecią liczbę całkowitą: "); // Zapytanie
17      int z = input.nextInt(); // Odczyt trzeciej liczby
```

```
18
19       int result = x * y * z; // Wyliczenie iloczynu
20
21       System.out.printf("Iloczyn wynosi %d%n", result);
22    } // Koniec metody main
23 } // Koniec klasy Product
```

```
Wpisz pierwszą liczbę całkowitą: 10
Wpisz drugą liczbę całkowitą: 20
Wpisz trzecią liczbę całkowitą: 30
```

Ćwiczenia

2.7. Wypełnij puste miejsca w następujących zdaniach:

a) _____ służy do dokumentowania programu i poprawy jego czytelności.

b) Decyzję w języku Java podejmuje się za pomocą _____.

c) Operatorami arytmetycznymi o takim samym priorytecie są mnożenie i _____.

d) Gdy nawiasy w wyrażeniu są zagnieżdżone, najpierw wykonywany jest _____ zestaw nawiasów.

e) Miejsce w pamięci, które w trakcie trwania programu może przechowywać różne wartości, nosi nazwę _____.

2.8. Napisz instrukcje Javy, które realizują opisane poniżej zadania:

a) Wyświetlenie tekstu `"Wpisz liczbę całkowitą: "` i pozostawienie kursora w tym samym wierszu.

b) Przypisanie wyniku mnożenia zmiennych `b` i `c` do zmiennej `a` typu `int`.

c) Użycie komentarza do wskazania, że program wylicza wynagrodzenie.

2.9. Określ, które z poniższych zdań są **prawdziwe**, a które **fałszywe**. Jeśli są fałszywe, wyjaśnij dlaczego.

a) Operatory Javy są wyliczane od lewej do prawej.

b) Wszystkie wymienione nazwy zmiennych są prawidłowe: `_pod_barem_`, `m928134`, `t5`, `j7`, `jej_sprzedaz$`, `jego_$bilans_laczny`, `a`, `b$`, `c`, `z` i `z2`.

c) Poprawne wyrażenie arytmetyczne w Javie bez nawiasów jest wyliczane od lewej do prawej.

d) Te wszystkie identyfikatory są uznawane za nieprawidłowe: `3g`, `87`, `67h2`, `h22` i `2h`.

2.10. Co wyświetli każda z poniższych instrukcji przy założeniu, że `x = 2` i `y = 3`?

a) `System.out.printf("x = %d%n", x);`

b) `System.out.printf("Wynikiem %d + %d jest %d%n", x, x, (x + x));`

c) `System.out.printf("x =");`

d) `System.out.printf("%d = %d%n", (x + y), (y + x));`

2.11. Które z poniższych instrukcji Javy zawierają zmienne o modyfikowanych wartościach?

a) `int p = i + j + k + 7;`

b) `System.out.println("zmienne, których wartości są modyfikowane");`

c) `System.out.println("a = 5");`

d) `int value = input.nextInt();`

2.12. Przy założeniu, że $y = ax^3 + 7$, które z poniższych instrukcji Javy odpowiadają temu równaniu?

a) `int y = a * x * x * x + 7;`
b) `int y = a * x * x * (x + 7);`
c) `int y = (a * x) * x * (x + 7);`
d) `int y = (a * x) * x * x + 7;`
e) `int y = a * (x * x * x) + 7;`
f) `int y = a * x * (x * x + 7);`

2.13. Określ kolejność wykonywania operacji w każdej z następujących instrukcji i wylicz wartość `x` po każdej operacji:

a) `int x = 7 + 3 * 6 / 2 - 1;`
b) `int x = 2 % 2 + 2 * 2 - 2 / 2;`
c) `int x = (3 * 9 * (3 + (9 * 3 / (3))));`

2.14. Napisz aplikację, która wyświetla w jednym wierszu liczby od 1 do 4, a każda liczba jest oddzielona od poprzedniej spacją. Użyj następujących technik:

a) pojedyncza instrukcja `System.out.println`;
b) cztery instrukcje `System.out.print`;
c) pojedyncza instrukcja `System.out.printf`.

2.15. **(Arytmetyka)** Napisz aplikację, która prosi użytkownika o wpisanie dwóch liczb całkowitych, pobiera je i wyświetla ich sumę, iloczyn, różnicę i iloraz. Użyj technik przedstawionych na rysunku 2.7.

2.16. **(Porównywanie liczb całkowitych)** Napisz aplikację, która prosi o dwie liczby całkowite, pobiera je od użytkownika i wyświetla większą spośród liczb z tekstem `"jest większa"`, natomiast jeśli liczby są sobie równe, wyświetla tekst `"Liczby są sobie równe"`. Użyj technik z rysunku 2.15.

2.17. **(Arytmetyka — najmniejszy i największy)** Napisz aplikację, która pobiera od użytkownika trzy liczby całkowite i wyświetla ich sumę, średnią, iloczyn, liczbę najmniejszą i największą. Użyj technik z rysunku 2.15. (**Uwaga:** Wyliczenie średniej w tym zadaniu powinno skutkować uzyskaniem średniej w postaci liczby całkowitej, czyli dla sumy wynoszącej 7 wynikiem powinno być 2, a nie 2,3333...).

2.18. **(Wyświetlanie kształtów z gwiazdek)** Napisz aplikację, która wyświetla prostokąt, elipsę, strzałkę i romb za pomocą znaków gwiazdki (*).

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

2.19. Co wyświetli poniższy kod?

```
System.out.printf("*%n**%n***%n****%n*****%n");
```

2.20. Co wyświetli poniższy kod?

```
System.out.println("*");
System.out.println("***");
System.out.println("*****");
System.out.println("****");
System.out.println("**");
```

2.21. Co wyświetli poniższy kod?

```
System.out.print("*");
System.out.print("***");
```

```
System.out.print("*****");
System.out.print("****");
System.out.println("**");
```

2.22. Co wyświetli poniższy kod?

```
System.out.print("*");
System.out.println("***");
System.out.println("*****");
System.out.print("****");
System.out.println("**");
```

2.23. Co wyświetli poniższy kod?

```
System.out.printf("%s%n%s%n%s%n", "*", "***", "*****");
```

2.24. **(Najmniejsza i największa liczba całkowita)** Napisz aplikację, która odczytuje pięć liczb całkowitych i wyświetla najmniejszą i największą z tej grupy. Wykorzystaj tylko techniki programistyczne poznane w tym rozdziale.

2.25. **(Liczby parzyste i nieparzyste)** Napisz aplikację, która odczytuje liczbę całkowitą i określa, czy jest to liczba parzysta czy nieparzysta. (Wskazówka: Użyj operatora reszty z dzielenia. Liczba całkowita to wielokrotność 2. Każda liczba podzielna przez 2 zwraca resztę równą 0).

2.26. **(Wielokrotność)** Napisz aplikację, która odczytuje dwie liczby całkowite i określa, czy pierwsza liczba jest wielokrotnością drugiej. (Wskazówka: Użyj operatora reszty z dzielenia).

2.27. **(Wzór szachownicy z gwiazdek)** Napisz aplikację, która wyświetli poniższy wzór:

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

2.28. **(Średnica, obwód i pole koła)** To ćwiczenie wybiega nieco w przód. W tym rozdziale omawialiśmy liczby całkowite i typ `int`. Java potrafi również reprezentować liczby zmiennoprzecinkowe definiujące część ułamkową, np. 3,14159. Napisz aplikację, która pobiera od użytkownika promień koła jako liczbę całkowitą, a następnie wylicza średnicę, obwód i pole koła, wykorzystując liczbę zmiennoprzecinkową `3.14159` jako π. Użyj technik z rysunku 2.7. **(Uwaga:** Możesz też użyć predefiniowanej stałej `Math.PI` dla wartości π. Stała ta jest dokładniejsza niż 3,14159. Klasa `Math` jest zdefiniowana w pakiecie `java.lang`, więc jest zaimportowana automatycznie i nie trzeba importować jej samodzielnie). Użyj następujących wzorów (*r* to promień):

$$\textit{średnica} = 2r$$
$$\textit{obwód} = 2\pi r$$
$$\textit{pole} = \pi r^2$$

Nie przechowuj wyników obliczeń w zmiennej. Użyj ich bezpośrednio jako argumentów instrukcji używającej metody `System.out.printf`. Wartościami obwodu i pola będą wartości zmiennoprzecinkowe. Użyj dla nich określnika formatu `%f`. Więcej informacji na ten temat znajdziesz w rozdziale 4.

2.29. **(Wartość całkowita znaku)** Oto kolejny wybieg w przyszłość. W tym rozdziale omawialiśmy liczby całkowite i typ `int`. Java potrafi również reprezentować duże i małe litery, a także znaki specjalne. Każdemu znakowi odpowiada liczba całkowita. To, jakiemu znakowi odpowiada konkretna liczba, zależy od tak zwanego zestawu znaków używanego przez komputer. Aby wskazać pojedynczy znak, wystarczy umieścić go między apostrofami, np. `'A'`.

Aby pobrać liczbę całkowitą odpowiadającą znakowi, można poprzedzić go konstrukcją `(int)`, czyli użyć zapisu:

```
(int) 'A'
```

Taki operator nazywamy operatorem rzutowania. (Więcej informacji na temat operatora rzutowania znajdziesz w rozdziale 3.). Poniższa instrukcja spowoduje wyświetlenie znaku i jego liczbowego odpowiednika:

```
System.out.printf("Znak %c ma wartość liczbową %d%n", 'A',
➥((int) 'A'));
```

Wykonanie kodu spowoduje wyświetlenie znaku `A` i wartości `65` (z zestawu znaków Unicode). Określnik formatu `%c` służy do wstawiania pojedynczych znaków, np. znaku `'A'`).

Korzystając z instrukcji podobnych do tych, które pojawiły się w rozdziale, napisz aplikację, która wyświetli liczbowe odpowiedniki niektórych liter, cyfr i symboli specjalnych. Wyświetl liczbowe wartości znaków `A B C a b c 0 1 2 $ * + /` oraz spacji.

2.30. **(Oddzielanie cyfr w liczbie całkowitej)** Napisz aplikację, która przyjmuje liczbę składającą się z pięciu cyfr, dzieli ją na poszczególne cyfry i wyświetla je oddzielone od siebie trzema spacjami. Jeśli użytkownik wpisze liczbę `42339`, program powinien wyświetlić tekst:

```
4   2   3   3   9
```

Załóż, że użytkownik wpisze liczbę o odpowiedniej liczbie cyfr. Co się stanie, jeśli liczba będzie miała więcej cyfr? Co się stanie, jeśli będzie ich miała mniej? (Wskazówka: Wykonanie tego zadania jest możliwe na podstawie wiedzy zdobytej w tym rozdziale. Trzeba użyć dzielenia i reszty z dzielenia do „wyłuskania" odpowiednich cyfr).

2.31. **(Tabela z kwadratami i sześcianami liczb)** Wykorzystując tylko i wyłącznie techniki poznane w tym rozdziale, napisz aplikację obliczającą kwadraty i sześciany liczb od 0 do 10 i wyświetlającą je w formie tabelarycznej jak poniżej:

```
liczba kwadrat sześcian
0      0       0
1      1       1
2      4       8
3      9       27
4      16      64
5      25      125
6      36      216
7      49      343
8      64      512
9      81      729
10     100     1000
```

2.32. **(Wartości dodatnie, ujemne i zero)** Napisz program przyjmujący pięć liczb, sprawdzający je i wyświetlający liczbę wartości dodatnich, ujemnych i wynoszących 0.

Uczynić świat lepszym

2.33. **(Kalkulator indeksu BMI)** Indeks masy ciała (BMI) wprowadziliśmy w ćwiczeniu 1.10. Wzór ma postać:

$$BMI = \frac{wagaWKilogramach}{wzrostWMetrach \times wzrostWMetrach}$$

Utwórz kalkulator BMI, który odczytuje od użytkownika wagę w kilogramach i wzrost w metrach, a następnie wylicza i wyświetla wartość BMI. Dodatkowo wyświetl informację o kategorii, do której został przypisany użytkownik na podstawie zakresów ze strony:

http://bmi-online.pl/

aby możliwe było łatwe określenie nadwagi lub otyłości.

(**Uwaga:** W tym rozdziale korzystaliśmy z typu `int` do reprezentacji liczb całkowitych. Ponieważ zastosowanie wagi w metrach powodowałoby bardzo niepoprawne wyniki bez części ułamkowej, warto poprosić użytkownika o wpisanie wagi w centymetrach i podzielić ją przez 100. W rozdziale 3. poznasz typ `double`, który umożliwia reprezentację liczb z ułamkami. Gdyby obliczenia wykonywać za pomocą tego typu, wynik również były liczbą zmiennoprzecinkową, czyli znacznie dokładniejszą).

2.34. **(Kalkulator przyrostu populacji na świecie)** Znajdź w internecie informację na temat aktualnej liczby ludzi na świecie i jej rocznego przyrostu. Napisz aplikację, która przyjmie te dwie wartości, a następnie wyświetli przewidywaną liczbę ludzi na świecie po jednym, dwóch, trzech, czterech i pięciu latach.

2.35. **(Kalkulator oszczędności dzięki carpoolingowi)** Odwiedź kilka stron WWW dotyczących tematu carpoolingu. Napisz aplikację, która oblicza koszt codziennych dojazdów, abyś mógł ustalić, ile pieniędzy zaoszczędziłbyś dzięki zmianie sposobu dojazdu na carpooling, który dodatkowo redukuje emisję gazów cieplarnianych i zmniejsza korki. Aplikacja powinna pobrać od użytkownika następujące informacje i wyświetlić dzienny koszt dojazdu do pracy:

a) łączna liczba kilometrów przejeżdżanych każdego dnia;

b) koszt litra paliwa;

c) średnie spalanie;

d) opłaty parkingowe w przeliczeniu na jeden dzień;

e) inne opłaty w przeliczeniu na jeden dzień.

Skorowidz

A

B

C

D

E

F

G

H

J

K

N

O

P

S

Ś

T

U

V

W

X

Z, Ź

Programuj profesjonalnie — ucz się od najlepszych!

Tworzenie oprogramowania to wspaniała umiejętność. Zdolny programista może w zasadzie pracować w dowolnej dziedzinie. Co więcej, szalony rozwój technologii informatycznych właściwie codziennie otwiera nowe rynki i nowe nisze. Niemal wszędzie potrzebne są procesor i oprogramowanie. I spora część tego cyfrowego tortu przypada programistom Javy. Najlepsze kąski jednak dostają się tym najlepszym, najzdolniejszym i najbardziej profesjonalnym.

Jeśli chcesz być właśnie takim programistą, wziąłeś do ręki właściwą książkę. Jest to klasyczny podręcznik, dzięki któremu wiele osób osiągnęło mistrzostwo w programowaniu w Javie. Zawarto tu wyjątkowo rzetelne, praktyczne i aktualne wprowadzenie do języka. W książce znajdziesz sporo informacji o nowej wersji języka — Javie 9 — oraz o świetnym narzędziu JShell, które ułatwia szybkie poznanie języka. Szczegółowo opisano JavaFX — najnowszy GUI i równocześnie zestaw narzędzi do nowych projektów. W przystępny i zrozumiały sposób przedstawiono dość trudne pojęcia, takie jak współbieżność, dzięki czemu bez problemu wykorzystasz moc systemów wielordzeniowych.

W tej książce między innymi:

- Solidne wprowadzenie do Javy — klasy, obiekty, metody
- Podstawy programowania, w tym programowania obiektowego
- Struktury danych, kolekcje, lambdy i strumienie
- Rozwiązania bazodanowe
- System modułów platformy Java 9

Paul Deitel jest czempionem Javy. Ma ponad 35 lat doświadczenia informatycznego. Ukończył MIT, otrzymał tytuły Java Certified Programmer i Java Certified Developer. Prowadził setki kursów programowania dla pracowników takich jednostek jak Cisco, IBM, Siemens, Sun Microsystems (obecnie Oracle), Dell, NASA, Boeing, Nortel Networks, Puma, iRobot.

Harvey Deitel ma ponad 55 lat doświadczenia informatycznego. Również ukończył MIT, a na Uniwersytecie Bostońskim zdobył stopień doktora matematyki. Uczył programowania na uczelniach, w urzędach, firmach i wojsku.

Publikacje Deitelów są rozpoznawane na całym świecie i były tłumaczone na dziesiątki języków.

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

ISBN 978-83-283-3973-6

Cena: 179,00 zł

Pearson