Wiesław Rychlicki

Programowanie w języku Java

Zbiór zadań z (p)odpowiedziami

Helion

Ebookpoint.pl kopia dla: pawher@o2.pl

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich.
Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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)

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

Materiały do książki można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/projaz.zip

ISBN: 978-83-246-6366-8

Copyright © Helion 2012

Printed in Poland.

- Poleć książkę na Facebook.com
- Kup w wersji papierowej
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Ebookpoint.pl kopia dla: pawher@o2.pl

Spis treści

Ebookpoint.pl kopia dla: pawher@o2.pl

Ebookpoint.pl kopia dla: pawher@o2.pl

Od autora

Niniejszy zbiór zadań jest propozycją dla rozpoczynających naukę programowania w języku Java. Jak sugeruje tytuł (może nieco dziwnie zbudowany), Czytelnik znajdzie tutaj zadania z zakresu programowania i *podpowiedzi* (wskazówki) dotyczące tego, jak zabrać się do ich rozwiązywania. Jeśli informacja zawarta w treści zadania i ewentualna podpowiedź (wskazówka, uwaga) okażą się niewystarczające, to można sięgnąć do *odpowiedzi*, gdzie zawarto dodatkowe wskazówki, istotne fragmenty rozwiązania lub kompletne listingi kodów źródłowych. Każdy dział jest poprzedzony krótkim wprowadzeniem teoretycznym, przykładem lub linkiem do odpowiedniego fragmentu dokumentacji języka zamieszczonej w internecie.

W początkowych rozdziałach umieszczono w tekście zadań sugerowane nazwy plików źródłowych (np. w zadaniu 2.3 zaproponowano nazwę pliku *Adres.java*). Rozwiązując zadanie, Czytelnik powinien pamiętać, że nazwa klasy musi być identyczna z nazwą pliku (bez rozszerzenia). Sugerowanie nazw plików (a przy tym nazw klas) ma na celu przyzwyczajenie Czytelnika do dobierania nazw klas odpowiednich do treści zadania. Ponadto różne nazwy plików (klas) umożliwiają gromadzenie rozwiązań kilku zadań w tym samym folderze. W dalszej części zbioru zaproponowano nazwy plików zgodne z numerami zadań, np. rozwiązanie zadania 13.8 zapisano w pliku *Z13_8.java*, a inny wariant rozwiązania tego zadania zapisano w pliku *Z13_8a.java*. Aby ułatwić Czytelnikowi poszukiwanie plików z rozwiązaniami, rozwiązanie na przykład zadania 2.3 (i innych początkowych zadań) zawarto dodatkowo w pliku *Z02_3.java*. Wszystkie pliki źródłowe umieszczono na serwerze *ftp://ftp.helion.pl/przyklady/projaz.zip*).

Ebookpoint.pl kopia dla: pawher@o2.pl

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 1.

Pierwszy krok — rozpoczynamy naukę programowania w języku Java

1. Historia Javy i pierwsze zadania

Zespół inżynierów z firmy *Sun*, pracując pod kierownictwem *Jamesa Goslinga*, rozpoczął w 1991 roku projekt pod nazwą *Green*. Celem projektu było utworzenie języka programowania dla niewielkich urządzeń elektronicznych. Język miał być niezależny od platformy sprzętowej. Doprowadziło to do utworzenia kodu pośredniego dla maszyny wirtualnej, który można uruchamiać w każdym urządzeniu wyposażonym w odpowiedni interpreter. Przez kilka lat projekt rozwijał się, ale produkt nie uzyskiwał pożądanej popularności. Język początkowo nazwano *Oak* (dąb), ale gdy okazało się, że ta nazwa jest już zajęta, zmieniono ją na *Java*.

Przełomem w rozwoju tego języka stało się napisanie w nim przeglądarki internetowej *HotJava*. Umożliwiała ona uruchamianie kodu wbudowanego w strony internetowe, czyli tzw. *apletów*. Efekty pracy opublikowano w maju 1995 roku w magazynie „Sun Word" i odtąd Java zaczęła zyskiwać popularność.

Na początku 1996 roku ukazała się pierwsza oficjalna wersja języka — *Java 1.0*. Hasłem promującym Javę było zdanie *Write Once, Run Anywhere* („Napisz raz, uruchamiaj wszędzie"). W kolejnych wersjach uzupełniano zauważone braki i dodawano nowe funkcjonalności. Systematycznie rosła liczba klas i interfejsów:

Ebookpoint.pl kopia dla: pawher@o2.pl

- **JDK[1] 1.0** (23 stycznia 1996 r. — 211 klas i interfejsów) — pierwsza oficjalna wersja Javy.
- **JDK 1.1** (19 lutego 1997 r. — 477 klas i interfejsów).
- **J2SE[2] 1.2** (8 grudnia 1998 r. — 1524 klasy i interfejsy) — nazwa kodowa *Playground*.
- **J2SE 1.3** (8 maja 2000 r. — 1840 klas i interfejsów) — nazwa kodowa *Kestrel*.
- **J2SE 1.4** (6 lutego 2002 r. — 2723 klasy i interfejsy) — nazwa kodowa *Merlin*.
- **J2SE 5.0[3]** (30 września 2004 r. — 3270 klas i interfejsów) — nazwa kodowa *Tiger*.
- **Java SE 6[4]** (11 grudnia 2006 r. — 3777 klas i interfejsów) — nazwa kodowa *Mustang*.
- **Java SE 7** (28 lipca 2011 r.) — nazwa kodowa *Dolphin*.

W 2007 roku firma Sun ogłosiła, że kolejne wersje języka będą dostępne na licencji GPL i taka jest właśnie licencja dla wersji Java SE 7. Autor podczas pracy nad zbiorem zadań używał wersji Java SE 6. Wszystkie przedstawione rozwiązania nie będą sprawiać problemów w Java SE 7. Od 27 stycznia 2010 roku nowym właścicielem Javy jest firma Oracle, która kupiła firmę Sun Microsystems.

Zadanie 1.1.

Przygotuj komputer do programowania w języku Java.

Poszukaj niezbędnych plików w internecie, wykorzystując np. frazę *Java download*, i zainstaluj je.

Zadanie 1.2.

Sprawdź, czy środowisko JDK do programowania w języku Java jest poprawnie zainstalowane na Twoim komputerze. Jaką wersję środowiska masz zainstalowaną?

---

[1] Java Development Kit — pakiet udostępniający kod bajtowy wszystkich klas standardowych, maszynę wirtualną do ich uruchamiania oraz źródła klas i narzędzia niezbędne do programowania w języku Java (m.in. kompilator, paker, debuger).

[2] Java 2 Platform, Standard Edition — podstawowa wersja platformy Java, pozwalająca tworzyć i uruchamiać aplikacje napisane w języku Java na komputerach stacjonarnych i serwerach. Od tego wydania Java rozwija się w trzech wersjach: J2SE, J2EE (Java 2 Platform, Enterprise Edition — serwerowa platforma programistyczna języka Java) i J2ME (Java 2 Platform Micro Edition — uproszczona wersja platformy Java, zaprojektowana z myślą o tworzeniu aplikacji mobilnych dla urządzeń o bardzo ograniczonych zasobach).

[3] Dotychczasową numerację 1.5 zmieniono na 5.0.

[4] J2SE zastąpiono Java SE, a z numeracji usunięto 0. Nadal jednak używa się również oznaczenia 1.6.

Ebookpoint.pl kopia dla: pawher@o2.pl

Sprawdź w oknie konsoli (praca w trybie MS-DOS) działanie poleceń `java` i `javac`.

Zadanie 1.3.

Przygotuj „ściągawkę" z instrukcją uruchamiania wirtualnej maszyny Javy (JVM).

Skorzystaj z opcji pomocy, jaką oferuje program *java.exe*.

Zadanie 1.4.

Przygotuj „ściągawkę" z instrukcją uruchamiania kompilatora Javy (*javac.exe*).

Metoda zastosowana w rozwiązaniu zadania 1.3 nie sprawdzi się w tym przypadku. Informacje są zawsze wyświetlane na ekranie. Należy poszukać innego rozwiązania.

Zadanie 1.5.

Przygotuj folder, w którym będziesz zapisywał swoje kody źródłowe, i kilka plików wsadowych ułatwiających dalszą pracę.

Na początek tak przygotowane narzędzia w zupełności nam wystarczą. Autor nie jest zwolennikiem strzelania z armaty do wróbli (generalnie uważa, że strzelanie do wróbli nie jest wskazane), wobec tego nie proponuje Czytelnikowi użycia skomplikowanych zintegrowanych środowisk programistycznych (IDE) i wieloplikowych projektów do kompilowania i uruchamiania kilku linijek kodu. Po wykonaniu tych zadań możesz kontynuować naukę programowania w języku Java.

2. JDK, Notatnik i klasyczny przykład „Hello World"

Ten przykład jest powielany w wielu podręcznikach i ma swoje odpowiedniki prawdopodobnie we wszystkich językach programowania.

Listing 1. *Hello.java*

```
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

Zadanie 2.1.

Przepisz dokładnie kod źródłowy podany na listingu 1. Skompiluj go i uruchom.

Ebookpoint.pl kopia dla: pawher@o2.pl

Skorzystaj z przygotowanych wcześniej plików wsadowych i dysku *X:*.

Zadanie 2.2.

Napisz polskojęzyczną wersję klasycznego przykładu z zadania 2.1. Klasę nazwij `Witam`; uruchomiona aplikacja powinna wyświetlić w konsoli napis (w dwóch wierszach):

```
Witaj, Świecie.
Uczę się programować w języku Java.
```

W systemie Windows kod źródłowy pisany jest w Notatniku (strona kodowa 1250), a efekty pracy programu wyświetlane są w konsoli (strona kodowa 852). Powoduje to nieprawidłowe wyświetlanie liter z polskimi znakami diakrytycznymi. Możliwe są dwa rozwiązania:

1. Zmiana strony kodowej w konsoli poleceniem: `chcp 1250`.
2. Uruchamianie aplikacji poleceniem: `java -Dfile.ecoding=CP852 NazwaKlasy`.

W obu przypadkach w konsoli powinna być ustawiona czcionka *TT Lucida Console*.

Zadanie 2.3.

Napisz program, który wyświetli na ekranie Twój adres w kilku wierszach, tak jak napisałbyś go na kopercie (plik źródłowy: *Adres.java*).

Użyj kilka razy instrukcji `System.out.println("...");`. Możesz również zapisać cały adres w jednym łańcuchu, wstawiając we właściwych miejscach znak specjalny `\n`, oznaczający przejście do nowego wiersza (ang. ***n**ew line*).

Zadanie 2.4.

Napisz program, który wyświetli na ekranie etykietkę zawierającą Twoje imię i nazwisko (plik źródłowy: *Etykieta.java*). Jako wzór (niekoniecznie do wiernego odtworzenia) przyjmij podany przykład:

```
******* Programowanie *******
*  obiektowe w języku Java  *
*         Jan Nowak         *
*****************************
```

W tym zadaniu i zadaniach podobnych wyświetlenie wiersza tekstu (z przejściem do następnego wiersza) możesz zrealizować na kilka sposobów, dających dokładnie taki sam rezultat:

```
System.out.println("Wiersz tekstu...");
System.out.print("Wiersz tekstu...\n");
System.out.print("Wiersz tekstu..."); System.out.println();
```

Element odpowiedzialny za przejście do nowego wiersza w kodzie wyróżniono pogrubieniem czcionki.

Ebookpoint.pl kopia dla: pawher@o2.pl

3. Znaki, tablice znaków i klasa String

Do reprezentacji znaków (liter, cyfr, znaków interpunkcyjnych i wielu innych symboli) służy typ `char`. Do zapisania jednego znaku potrzeba 16 bitów (dwa bajty) i stosowany jest standard Unicode.

Klasa `Character` opakowuje prosty typ `char` w obiekt; posiada jedno pole typu `char` i szereg metod działających na znakach.

Najczęściej posługujemy się stałymi znakowymi, np. `'A'` jest znakiem o kodzie *65*. W ten sposób możemy zapisywać wszystkie znaki widoczne na klawiaturze. Znaki specjalne posiadają symbole zastępcze: `\b` — backspace, `\t` — tabulator, `\n` — przejście do nowego wiersza, `\r` — powrót karetki (przejście na początek wiersza), `\"` — znak cudzysłowu, `\'` — znak apostrofu, `\\` — znak lewego ukośnika (ang. *slash*). Znaki Unicode możemy zapisywać w notacji szesnastkowej — wartości od `\u0000` do `\uFFFF` (65 536 znaków)[5]. W konsoli będą wyświetlone poprawnie znaki zdefiniowane w wybranej czcionce, pozostałe znaki zostaną zastąpione znakiem zapytania (`?`). Oto kilka operacji na zmiennych typu `char`:

```
char a;  // deklaracja zmiennej
a = 'A'; // przypisanie do zmiennej a znaku 'A' (stałej znakowej 'A')
char b = 'B'; // zadeklarowanie zmiennej b i zainicjowanie jej wartością 'B'
b = a; // przypisanie zmiennej b wartości zmiennej a
char alfa = '\u03B1'; // grecka litera alfa (α)
```

Przy okazji poznałeś komentarz liniowy (wierszowy) stosowany w kodach źródłowych. Tekst od znaków `//` do końca wiersza jest komentarzem (objaśnieniem dla programisty) i kompilator go ignoruje.

Zwykle pojedyncze znaki nam nie wystarczają, więc tworzymy tablice znaków. Zmienną tablicową (w tym przypadku tablicę znaków) możemy zadeklarować na dwa sposoby: `char[] znaki` lub `char znaki[]`. Słowo `znaki` jest identyfikatorem (nazwą) zmiennej, natomiast `char` nazwą typu danych przechowywanych w tablicy. Tablicami zawierającymi dane różnych typów będziemy posługiwali się wielokrotnie. Na razie zadeklarowaliśmy zmienną `znaki`. Teraz możemy utworzyć tablicę za pomocą operatora `new`:

```
znaki = new char[15];
```

Tablica `znaki` może przechowywać maksymalnie 15 elementów (taką wartość podaliśmy w chwili tworzenia tablicy) i są one indeksowane liczbami całkowitymi od 0 do 14 (o 1 mniej od rozmiaru tablicy). Mamy zatem dostęp do elementów tablicy za pośrednictwem zmiennych: `znaki[0]`, `znaki[1]`, …, `znaki[14]`.

Pozostaje wypełnić tablicę wartościami, np.: `znaki[0] = 'W'`, `znaki[1] = 'i'` itd. (pamiętajmy o zakresie indeksów; przekroczenie zakresu zakończy się zgłoszeniem wyjątku `ArrayIndexOutOfBounds`).

[5] Na tym etapie ta informacja w zupełności wystarczy. Obecnie standard Unicode umożliwia zakodowanie większej liczby znaków. Zastosowano siedemnaście tzw. przestrzeni numeracyjnych. My ograniczymy się do podstawowej przestrzeni numeracyjnej.

Ebookpoint.pl kopia dla: pawher@o2.pl

Tablicę znaków możemy również wypełnić wartościami (zainicjować) w trakcie definiowania:

```
char napis[] = {'W', 'i', 't', 'a', 'j'};
```

Rozmiar tej tablicy jest równy *5*, a indeksy elementów zawierają się w przedziale od *0* do *4*.

Tablica jest strukturą danych stanowiących zestaw elementów tego samego typu.

Mamy dwie równoważne deklaracje zmiennej tablicowej:

```
typ_danych[] nazwaTablicy;
typ_danych nazwaTablicy[];
```

Tablicę możemy zainicjować listą wartości umieszczoną w nawiasach { }. Rozmiar tablicy będzie równy liczbie podanych elementów:

```
typ_danych nazwaTablicy[] = {element_1, element_2, ..., element_n};
```

Tablicę możemy utworzyć za pomocą operatora `new`. Tworząc tablicę, należy określić liczbę elementów:

```
// zmienna nazwaTablicy była wcześniej zadeklarowana
nazwaTablicy = new typ_danych[liczba_elementów];
// deklaracja i tworzenie tablicy elementów
typ_danych[] nazwaTablicy = new typ_danych[liczba_elementów];
```

Rozmiar tablicy można odczytać za pomocą odwołania `nazwaTablicy.length`[6].

Indeksy elementów tablicy są liczbami całkowitymi i należą do przedziału od *0* do *n–1*, gdzie *n* jest liczbą elementów (rozmiarem) tablicy.

Zadanie 3.1.

Utwórz tablicę znaków zawierającą napis *Dzień dobry*. Napisz aplikację (plik źródłowy: *Witaj.java*), która wyświetli napis w konsoli.

> Parametrem wywołania metody `print()` lub `println()` (obiektu `out` z klasy `System`) może być wartość typu `char` lub `char[]`.

Zadanie 3.2.

Utwórz tablicę zawierającą znaki słowa *Informatyka*. Napisz program wyświetlający znaki zawarte w tablicy w następujący sposób (plik źródłowy: *Informatyka.java*):

a) pionowo — każdy znak w odrębnym wierszu,

b) poziomo — znaki rozdzielone dodatkowymi odstępami (tzw. *spacjowanie* lub *rozstrzelenie tekstu*).

[6] Tablica jest obiektem, który posiada pole (własność) `length`. Pojęcie to wkrótce będzie wyjaśnione dokładniej.

Ebookpoint.pl kopia dla: pawher@o2.pl

c) poziomo — wielkimi literami,

d) poziomo — małymi literami.


Wskazówka

Parametrem metody `print()` (`println()`) może być zmienna typu `char`. Przydatna okaże się również klasa `java.lang.Character` opakowująca typ prosty `char`. Obiekt typu `Character` posiada jedno pole typu `char` oraz szereg metod przetwarzających znaki (zob. *http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Character.html*). Zamiany wartości zmiennej `znak` typu `char` na wielką literę można dokonać z wykorzystaniem metody `Character.toUpperCase(znak)`. Zwrócony wynik (obiekt typu `Character`) może być parametrem metody `print()`. Zamiany na małe litery dokonamy przy użyciu metody `Character.toLowerCase(znak)`. Do pobierania znaków z tablicy można zastosować pętlę typu *for each*.

Zadanie 3.3.

Utwórz tablicę zawierającą znaki słowa *programowanie*. Napisz program zmieniający zawartość tablicy i wyświetlający efekty tych zmian (plik źródłowy: *Programowanie.java*):

a) zamień pierwszą literę na wielką,

b) zamień wszystkie litery na wielkie.


Wskazówka

Tym razem pętla typu `for each` nie wystarczy. Przeglądając tablicę, musimy mieć możliwość modyfikowania znaków. Aby zmodyfikować znak, należy znać jego indeks. Dostęp do wszystkich znaków w tablicy `dane` uzyskamy, stosując pętlę typu `for`:

```
for(int i=0; i < dane.length; ++i) {
// tu wykonaj przekształcenie związane z elementem dane[i]
}
```

Zadanie 3.4.

Utwórz tablicę zawierającą znaki słowa *programowanie*. Napisz program wyświetlający znaki zawarte w tablicy w kolejności odwrotnej — od końca do początku (plik źródłowy: *Wspak1.java*).


Wskazówka

Można zastosować pętlę `for`, w której następuje odliczanie w dół (od wartości większych do mniejszych `for(int i=dane.length-1; i >= 0; --i) ...`.


Uwaga

Do przeglądania (odczytu) zawartości tablicy możemy wykorzystać pętlę typ `for each`:

```
for(typ_danych z : tablica) instrukcja;
```

Zmienna `z` przyjmuje kolejno wartości wszystkich elementów tablicy (od początku do końca).

Do przeglądania (odczytu) lub modyfikowania (zapisu) zawartości tablicy możemy wykorzystać pętlę typu `for`:

```
for(int i = 0; i < tablica.length; ++i) instrukcja; /* Odliczanie w górę */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

lub:

```
for(int i = tablica.length-1; i >= 0; --i) instrukcja; /* Odliczanie w dół */
```

Zadanie 3.5.

Utwórz tablicę zawierającą znaki słowa *programowanie*. Napisz program odwracający kolejność znaków w tablicy (plik źródłowy: *Wspak2.java*).


Wskazówka

Przestawiaj kolejno pary elementów tablicy: pierwszy z ostatnim, drugi z przedostatnim — aż dojdziesz do środka tablicy.

Zadanie 3.6.

Przeanalizuj kod podany na listingu 2. Przepisz kod i uruchom tę aplikację. Sprawdź, czy wyniki na ekranie są zgodne z Twoimi oczekiwaniami. Na podstawie tego kodu i dokumentacji klasy `Character` napisz własną aplikację pokazującą działanie wybranej metody lub wartości zdefiniowanych stałych.

Listing 2. *DemoCharacter.java*

```
import static java.lang.System.*;
public class DemoCharacter {
    public static void main(String args[]) {
        /* Informacje o metodzie */
        out.println("Klasa: java.lang.Character");
        out.println("Metoda statyczna: digit\n");
        out.println("static int digit(int ch, int radix)");
        out.println("Returns the numeric value of the character ch
            in the specified radix.");
        out.println();
        /* Przykładowa tablica znaków */
        char znak[] = {'E', 'u', 'r', 'o', ' ', '2', '0', '1', '2'};
        /* Demonstracja działania metody */
        out.println("Wartość znaku jako cyfry w układzie dziesiątkowym
            (radix = 10)");
        for(char z : znak)
            out.println("Znak: "+z+"  Cyfra: "+Character.digit(z, 10));
        out.println("Uwaga: -1 oznacza, że znak nie jest cyfrą w tym
            układzie liczbowym.");
        out.println();
        out.println("Wartość znaku jako cyfry w układzie szesnastkowym
            (radix = 16)");
        for(char z : znak)
            out.println("Znak: "+z+"  Cyfra: "+Character.digit(z, 16));
        out.println("Uwaga: -1 oznacza, że znak nie jest cyfrą w tym
            układzie liczbowym.");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 3.7.

Napisz program, który utworzy dziesięcioelementową tablicę znaków i wypełni ją cyframi od 0 do 9 (plik źródłowy: *Cyfry.java*).

> Zadeklaruj tablicę znaków o odpowiednim wymiarze. Wykorzystaj tabelę kodów ASCII i możliwość zamiany kodu znaku na odpowiadający mu znak np. przez rzutowanie: `(char)65` zamienia liczbę *65* na znak o kodzie *65*, czyli wielką literę `'A'`.

Zadanie 3.8.

Napisz program, który utworzy szesnastoelementową tablicę znaków i wypełni ją cyframi układu szesnastkowego (plik źródłowy: *Cyfry16.java*).

> Możesz postąpić podobnie jak w zadaniu 3.7 lub wykorzystać statyczną metodę `forDigit()` z klasy `Character`.

4. Klasa String — operacje na tekstach

W Javie nie ma wbudowanego typu prostego do przechowywania ciągów (łańcuchów) znaków. Każdy łańcuch znaków jest obiektem klasy `String` należącej do standardowej biblioteki. Klasa `String` oferuje kilka konstruktorów i szereg metod (ponad 50) do przetwarzania łańcuchów. Literały łańcuchowe zapisujemy w cudzysłowie, np. `"Zadania z programowania"`. Ten literał jest obiektem i na jego przykładzie zostanie pokazanych kilka metod klasy `String`.

Zadanie 4.1.

Uruchom aplikację, której kod źródłowy przedstawiono na listingu 3. Na podstawie obserwacji wyników działania programu opisz działanie użytych metod klasy `String`. W przypadku wątpliwości zajrzyj do dokumentacji języka Java (np. *http://download.oracle.com/javase/6/docs/api/java/lang/String.html*).

Listing 3. *TestString.java*

```
public class TestString {
    public static void main(String args[]) {
        System.out.println("Zadania z programowania.");
        System.out.println("Zadania z programowania.".charAt(0));
        System.out.println("Zadania z programowania.".length());
        System.out.println("Zadania z programowania.".charAt(23));
        System.out.println("Zadania z programowania.".toUpperCase());
        System.out.println("Zadania z programowania.".toLowerCase());
        System.out.println("Zadania z programowania.".indexOf('z'));
        System.out.println("Zadania z programowania.".indexOf("prog"));
        System.out.println("Zadania z programowania.".replace('.', '?'));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Zadania z programowania.".
            replace("adania", "dania"));
        System.out.println("Zadania z programowania.".
            replaceAll("ania", "anka"));
        System.out.println("Zadania z programowania.".
            replaceFirst("ania", "anka"));
        System.out.println("Zadania z programowania.".substring(10));
        System.out.println("Zadania z programowania.".substring(10, 17));
        System.out.println("Zadania z programowania.".
            concat("\b z podpowiedziami."));
        System.out.println("Zadania z programowania."+"\b"+
            " z odpowiedziami.");
    }
}
```

Oczywiście możemy tworzyć zmienne typu obiektowego `String`. Ponieważ raz utworzony łańcuch znaków (ang. *string*) nie może być zmieniany, to każda operacja powoduje tworzenie nowego łańcucha. Zmienna jest referencją do łańcucha znaków, więc możemy przypisać jej inny łańcuch (w szczególności zmieniony obraz łańcucha, który wskazywała wcześniej). Nie powoduje to problemów z pamięcią, gdyż zbędne obiekty w Javie są automatycznie usuwane. Utworzoną zmienną należy zainicjować:

```
String napis = "Zadania z programowania.";
```

lub:

```
String napis = new String("Zadania z programowania.");
```

Łańcuch możemy również utworzyć z tablicy znaków, np.:

```
char[] znaki = {'z', 'a', 'd', 'a', 'n', 'i', 'e'};
String napis = new String(znaki);
```

Zadanie 4.2.

Napisz program z listingu 3., wykorzystując zamiast literałów zmienne reprezentujące obiekty typu `String` lub zmienne typu `char`. Klasę nazwij `DemoString`.


Wskazówka

> Aby kod był czytelny, wprowadź odpowiednie nazwy zmiennych, np. `char kropka = '.';` itp.

Rozwiążemy teraz kilka zadań podobnych do zadań z rozdziału 3. Zastąpimy tablicę znaków odpowiednim łańcuchem. Należy jednak pamiętać, że pomimo jednakowej zawartości te obiekty się różnią. Możemy dokonywać konwersji w obie strony pomiędzy tablicą i łańcuchem znaków.

Zadanie 4.3.

Utwórz łańcuch zawierający napis *Dzień dobry*. Napisz aplikację (plik źródłowy: *WitajStr.java*), która wyświetli napis w konsoli:

a) pionowo — każdy znak w odrębnym wierszu,

Ebookpoint.pl kopia dla: pawher@o2.pl

b) poziomo — znaki rozdzielone dodatkowymi odstępami (tzw. spacjowanie lub rozstrzelenie tekstu),

c) poziomo — wielkimi literami,

d) poziomo — małymi literami.


Wskazówka

W podpunktach a) i b) możesz skorzystać z pętli typu *for* i metody `charAt()`. Można też dokonać konwersji łańcucha na tablicę znaków (metoda `toCharArray()` z klasy `String`) i dalej postępować według rozwiązania zadania 3.2.

Zadanie 4.4.

Utwórz łańcuch znaków zawierający słowo *programowanie*. Napisz program zmieniający zawartość łańcucha i wyświetlający efekty tych zmian (plik źródłowy: *ProgramowanieStr.java*):

a) zamień pierwszą literę na wielką,

b) zamień wszystkie litery na wielkie.


Wskazówka

Możemy utworzyć nowy łańcuch, spełniający warunki zadania, i przypisać jego wartość tej samej zmiennej. Dotychczasowa wartość zostanie usunięta z pamięci w chwili, gdy nie będzie już dalej potrzebna.

Zadanie 4.5.

Utwórz łańcuch znaków zawierający słowo *programowanie*. Napisz program, który znaki zawarte w łańcuchu będzie wyświetlał w kolejności odwrotnej, od końca do początku (plik źródłowy: *WspakStr1.java*).

Zadanie 4.6.

Utwórz łańcuch znaków zawierający słowo *programowanie*. Napisz program odwracający kolejność znaków w łańcuchu (plik źródłowy: *WspakStr2.java*).


Wskazówka

Zob. wskazówkę do zadania 4.4.

Zadanie 4.7.

Napisz program, który utworzy łańcuch znaków wypełniony cyframi od 0 do 9 (plik źródłowy: *CyfryStr.java*).


Uwaga

Rozwiązanie tego zadania jest zupełnie banalne — poprawna odpowiedź to: `String cyfry = "0123456789";`. Spróbuj jednak zbudować ten łańcuch z poszczególnych znaków (zob. zadanie 3.7). Takie podejście do zagadnienia może w przyszłości się przydać.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 4.8.

Napisz program, który utworzy łańcuch znaków wypełniony cyframi układu szesnastkowego (plik źródłowy: *CyfryStr16.java*).


Uwaga

Zob. wskazówkę do zadania 4.7.

5. Tablica argumentów aplikacji

W nagłówku metody `main()`jest zadeklarowana tablica argumentów `String args[]` lub `String[] args`.

W tablicy zawarte są referencje do obiektów typu `String` przekazanych przez system operacyjny podczas uruchomienia aplikacji poleceniem:

```
java NazwaKlasy parametr1 parametr2 ... parametrN
```

Pole `args.length` zawiera liczbę przekazanych argumentów. Argumenty w programie są dostępne przy użyciu indeksów od `0` do `args.length-1`: `args[0]`, `args[1]`, …, `args[N-1]`, gdzie `N` oznacza liczbę argumentów.

Jeśli aplikację wywołamy bez dodatkowych parametrów (`java NazwaKlasy`), czyli liczba argumentów będzie równa `0`, to wtedy tablica `args` będzie pusta (nie będzie zawierać żadnego elementu).

W kolejnych zadaniach wykorzystamy tablicę argumentów do przekazywania danych do aplikacji.

Zadanie 5.1.

Napisz program, który wyświetli na ekranie liczbę argumentów wywołania aplikacji oraz podane argumenty. Każdy argument powinien być wyświetlony w odrębnym wierszu (plik źródłowy: *Argumenty.java*).


Wskazówka

Do przeglądania zawartości tablicy wykorzystaj pętle typu `for` lub `for each` (zob. zadanie 3.2).

Zadanie 5.2.

Napisz program (plik źródłowy: *Osoba.java*), który uruchamiany z dwoma parametrami, `imię` i `nazwisko`, wyświetli na ekranie w kolejnych wierszach te dane według schematu:

```
Nazwisko: Kowalska
Imię: Maria
Nazwisko i imię: KOWALSKA Maria
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
Inicjały: MK
Login: KOmar
```

Wielkości liter na wydruku powinny być zgodne z przykładem, niezależnie od tego, jakie wielkości liter wykorzysta użytkownik, podając imię i nazwisko.

Wypisz błędy, które powstaną podczas uruchomienia aplikacji z niewłaściwą liczbą parametrów.

Zadanie 5.3.

Napisz program (plik źródłowy: *ArgsWspak.java*), który uruchomiony z kilkoma argumentami wypisze listę argumentów, oddzielonych odstępami, w odwrotnej kolejności.

Zadanie 5.4.

Napisz program (plik źródłowy: *ArgWspak.java*), który uruchomiony z kilkoma argumentami wypisze każdy argument w odrębnym wierszu, odwracając przy tym kolejność znaków w argumencie.


Wskazówka

Należy użyć dwóch pętli, stosując tzw. *zagnieżdżanie pętli*. Zewnętrzna pętla powinna odczytać kolejno wszystkie argumenty, a wpisana w jej ciele wewnętrzna pętla ma za zadanie odwrócić kolejność znaków w bieżącym argumencie.

Zadanie 5.5.

Napisz program, który wyświetli na ekranie etykietkę zawierającą imię i nazwisko podane jako parametry wywołania aplikacji (plik źródłowy: *EtykietaArg.java*). Zmodyfikuj odpowiednio rozwiązanie zadania 2.4.


Wskazówka

Oblicz długość tekstu (imienia i nazwiska razem z odstępem między nimi), liczbę odstępów pomiędzy między znakiem * a tekstem oraz po tekście przed znakiem *. Stosowną liczbę odstępów możesz wyświetlić w pętli lub wyciąć jako podłańcuch z łańcucha złożonego z określonej liczby odstępów. Ze względu na konieczność obliczeń zadanie wyprzedza nieco omawianą teorię.

6. Prawda czy fałsz — logiczny typ danych

Prosty typ danych `boolean` może przyjmować jedną z dwóch wartości: `false` (*fałsz*) i `true` (*prawda*). Te dwie wartości służą do oceny logicznej różnych zdań (warunków) i są podstawą do działania instrukcji sterujących przebiegiem programu (instrukcji warunkowych, instrukcji wyboru i pętli).

Klasa `Boolean` opakowuje prosty typ `boolean`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 6.1.

Napisz program, który w formie tabeli przedstawi działanie operatorów logicznych (plik źródłowy: *OperatoryLogiczne.java*).


Wskazówka

Możesz zbudować dwuelementową tablicę wartości logicznych i użyć pętli typu `for each` do pobierania wartości z tej tablicy. Dzięki zagnieżdżeniu pętli możesz w łatwy sposób przetestować wszystkie wartości dla dwóch i większej liczby zdań logicznych. Do dyspozycji masz trzy operatory:

- `!` — negację (NOT),
- `&` lub `&&` — koniunkcję (iloczyn logiczny, AND),
- `|` lub `||` — alternatywę (sumę logiczną, OR).

Zadanie 6.2.

Napisz program (plik źródłowy: *PrawaLogiczne.java*), który w formie tabeli przedstawi dowód następujących praw logicznych:

a) prawo wyłączonego środka $p \vee \neg p$,

b) prawo niesprzeczności $\neg(p \wedge \neg p)$,

c) prawo podwójnego przeczenia $\neg(\neg p) \Leftrightarrow p$.


Uwaga

W zapisie matematycznym wyrażeń logicznych zastosowano następujące symbole: $\neg$ — negacja, $\wedge$ — koniunkcja, $\vee$ — alternatywa, $\Leftrightarrow$ — równoważność. Równoważność możemy w programie zastąpić operatorem *jest równe* (`==` — dwa znaki równości obok siebie). Wynikiem porównania dwóch wartości należących do typu prostego jest wartość logiczna.

Zadanie 6.3.

Napisz program (plik źródłowy: *PrawaDeMorgana.java*), który w formie tabeli przedstawi dowód praw De Morgana:

a) I prawo De Morgana — prawo zaprzeczenia koniunkcji
$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$,

b) II prawo De Morgana — prawo zaprzeczenia alternatywy
$\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$.

Zadanie 6.4.

Napisz program, który przedstawi tabelę wartości logicznych implikacji (plik źródłowy: *Implikacja.java*).

Ebookpoint.pl kopia dla: pawher@o2.pl

Przyjmijmy, że wartości logiczne są uporządkowane i `false` poprzedza `true`. Implikacja jest fałszywa tylko w jednym przypadku `true` $\Rightarrow$ `false` (z prawdziwych założeń nie można udowodnić tezy fałszywej), w pozostałych przypadkach jest prawdziwa (z prawdy wynika prawda, na podstawie fałszywych założeń można udowodnić zarówno *prawdę*, jak i *fałsz*). Zauważmy, że stosując operator *mniejsze lub równe* (<=) dla dwóch wartości logicznych, uzyskamy taki sam wynik jak w przypadku implikacji tych zdań. W Javie operator <= nie został zdefiniowany dla typu `boolean`. Możemy skorzystać z metod klasy opakowującej i porównania rezultatu z liczbą 1: `Boolean.valueOf(p).compareTo(q) < 1`. Analizę tego wyrażenia pozostawiamy Czytelnikowi.

Zadanie 6.5.

Napisz program, który przedstawi tabelę wartości logicznych alternatywy wykluczającej (plik źródłowy: *Xor.java*).

Alternatywa wykluczająca (*p albo q*) jest prawdziwa, gdy jedno ze zdań jest prawdziwe, a drugie jest fałszywe, czyli ich wartości logiczne nie są równe.

Zadanie 6.6.

Napisz programy (pliki źródłowe: *PrawoLogiczneX.java*, gdzie *X* oznacza jedną z liter *A*, *B*, *C*, *D* lub *E*, zgodną z podpunktem rozwiązywanego zadania) przedstawiające w formie tabeli dowody następujących praw logicznych:

a) prawo przechodniości implikacji $[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$,

b) prawo rozdzielności alternatywy względem koniunkcji $[p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$,

c) prawo rozdzielności koniunkcji względem alternatywy $[p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$,

d) prawo odrywania $[(p \Rightarrow q) \wedge q] \Rightarrow q$,

e) prawo eliminacji implikacji $(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$.

W prawach zawierających implikację (podpunkty a), d) i e)) skorzystaj z metody statycznej `impl()`, zdefiniowanej w rozwiązaniu zadania 6.4. Uprości to w istotny sposób zapis zadania. Można też wprowadzić dodatkowe zmienne typu `boolean`, przechowujące wartości wyrażeń stojących po lewej i prawej stronie badanych równoważności.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 6.7.

W klasie `String` (od wersji Javy 1.6) dostępna jest metoda `isEmpty()`, zwracająca wartość `true`, gdy długość łańcucha znaków jest równa `0`, oraz wartość `false` w pozostałych przypadkach. Napisz prostą aplikację (plik źródłowy: *StringIsEmpty.java*) pokazującą działanie tej metody.


Wskazówka

> Ustal wartość łańcucha, a następnie wyświetl na ekranie ten łańcuch, jego długość i wartość zwróconą przez metodę `isEmpty()`.

Zadanie 6.8.

Napisz program (plik źródłowy: *TestChar.java*) demonstrujący działanie metod `isDigit()`, `isLetter()`, `isLetterOrDigit()`, `isLowerCase()`, `isSpaceChar()`, `isUpperCase()` i `isWhiteSpace()`. Wyniki przedstaw w postaci tabeli. Zadanie wykonaj dla zestawu znaków zapisanego w łańcuchu:

a) `"A\240b3&4\040"`,

b) `"Łoś_0+\t"`,

c) `"#\"\304\\\344\b\n"`.

7. Liczby całkowite typu int i klasa Integer

Prosty typ danych `int` pozwala na przechowywanie czterobajtowych (32-bitowych) liczb całkowitych ze znakiem.

Klasa `Integer` opakowuje prosty typ `int` i oferuje dwa konstruktory budujące obiekt na podstawie liczby całkowitej lub łańcucha znaków. Zawiera m.in. szereg metod przeznaczonych do konwersji obiektów klasy `Integer` na inne typy liczbowe lub łańcuchy znaków.

Zadanie 7.1.

Uruchom aplikację, której kod źródłowy przedstawiono na listingu 4. Na podstawie obserwacji wyników działania programu opisz przeznaczenie użytych stałych i metod statycznych klasy `Integer`. W kodzie źródłowym programu, w miejscu trzech kropek, wpisz odpowiednie teksty objaśniające wyświetlane wyniki. W przypadku wątpliwości zajrzyj do dokumentacji języka Java (np. *http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Integer.html*).

Listing 4. *StaticInteger.java*

```
public class StaticInteger {
    public static void main(String args[]) {
        System.out.println("Wybrane stałe i metody statyczne klasy Integer\n");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("...: "+Integer.MIN_VALUE);
        System.out.println("...: "+Integer.MAX_VALUE);
        System.out.println("...: "+Integer.SIZE);
        int a = 179;
        System.out.println("a = "+a);
        System.out.println("...: "+Integer.toBinaryString(a));
        System.out.println("...: "+Integer.toOctalString(a));
        System.out.println("...: "+Integer.toHexString(a));
        System.out.println("...: "+Integer.toString(a));
        System.out.println("...: "+Integer.toString(a, 4));
        int b = Integer.parseInt("-177");
        System.out.println("b = "+b);
        int c = Integer.parseInt("1000", 8);
        System.out.println("c = "+c);
        System.out.println("...: "+Integer.signum(a));
        System.out.println("...: "+Integer.signum(b));
        System.out.println("...: "+Integer.signum(0));
    }
}
```

Zadanie 7.2.

Uruchom aplikację przedstawioną na listingu 5. Na podstawie uzyskanych wyników i dokumentacji objaśnij (w postaci komentarza) użyte metody klasy `Integer`.

Listing 5. *ObjectInteger.java*

```
public class ObjectInteger {
    public static void main(String args[]) {
        System.out.println("Wybrane metody obiektów klasy Integer\n");
        /* tworzenie obiektów */
        Integer a = new Integer(1024);
        Integer b = new Integer("02000");
        Integer c = Integer.decode("02000");
        Integer d = Integer.decode("0x2000");
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("c = "+c);
        System.out.println("d = "+d);
        /* porównania obiektów */
        System.out.println("...? "+a.equals(b));
        System.out.println("...? "+a.equals(c));
        System.out.println("...? "+a.compareTo(c));
        System.out.println("...? "+c.compareTo(d));
        System.out.println("...? "+d.compareTo(c));
        /* zmiana wartości obiektu */
        a = Integer.valueOf(1000);
        b = Integer.valueOf("1000");
        c = Integer.valueOf("1000", 2);
        d = Integer.valueOf("1000", 16);
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("c = "+c);
        System.out.println("d = "+d);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zmienne i stałe (literały) typu `int` mogą być wykorzystane do wykonywania obliczeń w zbiorze liczb całkowitych, w zakresie ograniczonym 32-bitowym rozmiarem typu (od -2^{31} do $2^{31}-1$). Do dyspozycji mamy cztery podstawowe operatory: dodawanie +, odejmowanie -, mnożenie * i dzielenie (całkowite) / oraz operator %, służący do obliczania reszty z dzielenia dwóch liczb całkowitych.

Wartość całkowitą (potrzebną do obliczeń) zawartą w obiekcie możemy uzyskać przy użyciu metody `intValue()`.

Zadanie 7.3.

Dla dwóch dowolnie wybranych liczb całkowitych typu `int` napisz aplikację pokazującą działanie operatora + (plik źródłowy: *Dodawanie.java*). Wynik wyświetlaj w postaci:

```
a = 3, b = 15
a + b = 18
```

> Zadeklaruj i zainicjuj dwie zmienne typu `int`, np. `int a = 2451, b = 375;`. Wyrażenia wykonujące obliczenia możesz umieścić jako parametry w metodzie `println()`.

Zwróć uwagę na kolejność wykonywania działań, gdy operatora + używasz do obliczeń i łączenia tekstów!

Zadanie 7.4.

Dla dwóch dowolnie wybranych liczb całkowitych typu `int` napisz aplikacje pokazujące działanie pozostałych operatorów arytmetycznych:

a) odejmowanie - (plik źródłowy: *Odejmowanie.java*),

b) mnożenie * (plik źródłowy: *Mnożenie.java*),

c) dzielenie / (plik źródłowy: *Dzielenie.java*),

d) reszta z dzielenia % (plik źródłowy: *Reszta.java*).

Wyniki wyświetlaj podobnie jak w zadaniu 7.3.

Zadanie 7.5.

Napisz aplikację (plik źródłowy: *Dzielenie2.java*) demonstrującą dzielenie z resztą dla liczb całkowitych o różnych znakach. Wyniki wyświetlaj w postaci znanej z lekcji matematyki, np. `13 : 2 = 6 r. 1`.

Zadanie 7.6.

Napisz aplikację (plik źródłowy: *Suma.java*), która obliczy sumę dwóch liczb naturalnych podanych jako parametry wywołania. Wynik wyświetl na ekranie.

> Parametry są przekazywane do programu jako łańcuchy znaków. Potrzebna będzie zamiana łańcucha na liczbę całkowitą. Można użyć odpowiedniej metody statycznej lub posłużyć się obiektami klasy `Integer`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 7.7.

Napisz aplikację (plik źródłowy: *Sumuj.java*), która obliczy sumę serii liczb naturalnych podanych jako parametry wywołania aplikacji. Wynik wyświetl na ekranie.

> Wprowadź zmienną `suma` typu `int` i zainicjuj ją wartością `0`. Korzystając z pętli typu *for each*, przeglądaj tablicę argumentów, zamieniaj argument (łańcuch znaków) na liczbę całkowitą i dodawaj ją do sumy. Po przejrzeniu całej tablicy wyświetl wynik.

Zadanie 7.8.

Napisz aplikację (plik źródłowy: *Zamiana.java*), która na podstawie dwóch parametrów całkowitych — podstawy systemu liczbowego od *2* do *36* i liczby dziesiętnej — wyświetli na ekranie liczbę zapisaną we wskazanym systemie według podanego wzoru: `102[10] = 123[9]`.

Zadanie 7.9.

Napisz aplikację (plik źródłowy: *Zamiana2.java*), która na podstawie dwóch parametrów — podstawy systemu liczbowego od *2* do *36* i liczby zapisanej w systemie o podstawie podanej jako pierwszy parametr — wyświetli na ekranie tę liczbę zapisaną w systemie dziesiętnym według podanego wzoru: `123[9] = 102[10]`.

8. Inne typy liczb całkowitych w Javie

Do zapisywania liczb całkowitych w pamięci komputera najczęściej używamy 1, 2, 4 lub 8 bajtów (odpowiednio 8, 16, 32 lub 64 bitów). Odpowiada to następującym typom liczb całkowitych ze znakiem w języku Java: `byte`, `short`, `int` lub `long`.

Z każdym z wymienionych typów prostych związana jest klasa opakowująca — `Byte`, `Short`, `Integer` lub `Long`. Klasy te dziedziczą z klasy abstrakcyjnej `java.lang.Number`. Zmianie ulegają jedynie zakresy dostępnych wartości, natomiast metody mają identyczne nazwy i analogiczny sposób działania.

Każda z wymienionych klas posiada metody zwracające wartość reprezentowaną przez obiekt jako wartość jednego z pozostałych typów całkowitych.

Zadanie 8.1.

Przepisz i uruchom kod źródłowy przedstawiony na listingu 6. Opisz krótko zastosowane metody. Uzasadnij otrzymane rezultaty. Powtórz ćwiczenie, zmieniając wartość liczby `n` na minimalną dostępną wartość typu `long`.

Listing 6. *LongNumber.java*

```
public class LongNumber {
    public static void main(String args[]) {
        long n = Long.MAX_VALUE;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("n = "+n);
        System.out.println("BIN: "+Long.toBinaryString(n));
        System.out.println("HEX: "+Long.toHexString(n));

        long m = n+1;
        System.out.println("m = "+m);
        System.out.println("BIN: "+Long.toBinaryString(m));
        System.out.println("HEX: "+Long.toHexString(m));

        Long max = new Long(n);
        System.out.println("Zamiana na typ int, m = "+max.intValue());
        System.out.println("BIN: "+
            Integer.toBinaryString(max.intValue()));
        System.out.println("HEX: "+Integer.toHexString(max.intValue()));

        System.out.println("Zamiana na typ int, m = "+(int)m);
        System.out.println("BIN: "+Integer.toBinaryString((int)m));
        System.out.println("HEX: "+Integer.toHexString((int)m));
    }
}
```

Zadanie 8.2.

Napisz program (plik źródłowy: *MinMax.java*) wyświetlający minimalne i maksymalne wartości liczb całkowitych typu `byte`, `short`, `int` i `long`. Wyniki wyświetl w postaci: `nazwa typu <wartość minimalna, wartość maksymalna>`.

Wykorzystaj stałe `MIN_VALUE` i `MAX_VALUE` z klas opakowujących typy proste.

Zadanie 8.3.

Napisz program (plik źródłowy: *MaxPositive.java*) wyświetlający maksymalne wartości liczb całkowitych typu `byte`, `short`, `int` i `long`. Wyniki wyświetl w postaci binarnej i szesnastkowej.

Klasy opakowujące `Integer` i `Long` zawierają metody `toBinaryString()` i `toHexString()` — tych metod nie ma w klasach `Byte` i `Short`.

Zadanie 8.4.

Napisz program (plik źródłowy: *Kodowanie.java*) zamieniający łańcuch znaków (np. słowo *kodowanie*) na ciąg bajtów odpowiadających tym znakom.

Poszukaj odpowiedniej metody w klasie `String`.

Zadanie 8.5.

Napisz program (plik źródłowy: *Dekodowanie.java*) zamieniający ciąg bajtów (np. `{115, 122, 121, 102, 114}`) na łańcuch znaków odpowiadających tym liczbom.

Ebookpoint.pl kopia dla: pawher@o2.pl

Jeden z konstruktorów klasy `String` tworzy łańcuch znaków na podstawie tablicy bajtów.

Zadanie 8.6.

Napisz program (plik źródłowy: *DodajCyfry.java*) obliczający sumę cyfr liczby całkowitej dodatniej podanej:

a) w postaci tekstu — łańcucha znaków (cyfr) reprezentujących tę liczbę,

b) jako wartość typu `long`.

Skorzystaj z rozwiązania zadania 8.4 i pamiętaj, że kody cyfr są równe: `0` — 48, `1` — 49, `2` — 51 itd. Liczbę możesz zamienić na łańcuch znaków.

Zadanie 8.7.

Napisz program (plik źródłowy: *Txt2Hex.java*) zamieniający łańcuch znaków ASCII na łańcuch cyfr szesnastkowych odpowiadających tym znakom.

Zamień tekst na ciąg bajtów, a następnie przedstaw bajty w postaci szesnastkowej. Na jeden bajt (znak *ASCII*) przypadną dwie cyfry szesnastkowe.

Zadanie 8.8.

Napisz program (plik źródłowy: *Hex2Txt.java*) zamieniający łańcuch cyfr szesnastkowych na odpowiadający mu łańcuch znaków ASCII (dwie cyfry przypadają na jeden znak).

9. Typy liczb zmiennoprzecinkowych

Typy zmiennoprzecinkowe służą do przechowywania liczb z częścią ułamkową. Java oferuje dwa typy zmiennoprzecinkowe zgodne ze standardem *IEEE 754*:

- `float` — dane zmiennoprzecinkowe *32*-bitowe pojedynczej precyzji o dokładności *7 – 8* cyfr znaczących,
- `double` — dane zmiennoprzecinkowe *64*-bitowe podwójnej precyzji o dokładności *15* cyfr znaczących.

Zadanie 9.1.

Przepisz i uruchom kod źródłowy przedstawiony na listingu 7. Zobacz efekty działania programu. Czy na podstawie zapisu kodu źródłowego (bez czytania dokumentacji) możesz określić działanie użytych metod? Sporządź krótki opis kodu i efektów jego działania.

Ebookpoint.pl kopia dla: pawher@o2.pl

Listing 7. *DemoDouble.java*

```
public class DemoDouble {
    public static void main(String[] args) {
        double a = Double.MIN_VALUE;
        System.out.println("Minimalna wartość dodatnia: "+a);
        System.out.println("BIN: "+Long.toBinaryString(Double.
            doubleToLongBits(a)));
        System.out.println("HEX: "+Double.toHexString(a));
        a = -a;
        System.out.println("Liczba przeciwna: "+a);
        System.out.println("BIN: "+Long.toBinaryString(Double.
            doubleToLongBits(a)));
        System.out.println("HEX: "+Double.toHexString(a));
    }
}
```

Zadanie 9.2.

Przedstaw w postaci słowa 64-bitowego reprezentację binarną kilku wybranych liczb typu `double` (plik źródłowy: *DoubleBin.java*). Z ciągu 64 bitów wyodrębnij *bit znaku*, *cechę* i *mantysę*.

a) 0,25; 0,5; 1,0; 2,0; 512,0;

b) 0,01; 0,1; 1,0; 10,0; 100,0;

c) 1,367; 1367; 13,67; –13,67; 1,367e–12; 1,367e12;

d) symbole specjalne: `Double.NaN` (ang. *Not-a-Number*), np. wynik pierwiastkowania liczby ujemnej, `Double.NEGATIVE_INFINITY` ($-\infty$ — ujemna nieskończoność, np. wynik dzielenia –1/0,0), `Double.POSITIVE_INFINITY` ($+\infty$ — dodatnia nieskończoność, np. wynik dzielenia 1/0,0).

e) dwie reprezentacje zmiennoprzecinkowe zera `0.0` i `-0.0` oraz wartości ekstremalne `Double.MIN_VALUE`, `Double.MAX_VALUE` i `2.2250738585072014e-308`.


Wskazówka

Możemy zastosować metodę statyczną `doubleToLongBits()` z klasy `Double` i zamienić liczbę zmiennoprzecinkową typu `double` na liczbę całkowitą typu `long` o identycznej reprezentacji binarnej. W dalszej kolejności zamienimy liczbę typu `long` na odpowiadający jej ciąg cyfr binarnych. Niestety, metoda statyczna `toBinaryString()` z klasy `Long`, tworząc ciąg cyfr binarnych odpowiadających liczbie typu `long`, pomija zera nieznaczące. Uzupełnimy te zera do 64 znaków w łańcuchu, wycinając je jako podciąg o odpowiedniej długości z ciągu (łańcucha) złożonego z 64 cyfr `0` i dodając do ciągu zer uzyskaną postać binarną liczby.

Musimy pamiętać, że indeksy znaków w łańcuchu są liczone od strony lewej do prawej, natomiast bity w liczbie binarnej są numerowane od strony prawej (najmłodszy bit to 0) do lewej (najstarszy bit to 63).

Znaczenie poszczególnych bitów w reprezentacji binarnej liczby typu `double` jest następujące: bit nr 63 — *bit znaku*, bity o numerach od 52 do 62 — *cecha* liczby (11 bitów), bity o numerach od 0 do 51 (52 bity) — część ułamkowa *mantysy*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 9.3.

Ciąg bitów 11111000001111000011100011001O reprezentuje pewną liczbę zmiennoprzecinkową pojedynczej precyzji (zera nieznaczące pominięto). Jaką wartość dziesiętną ma ta liczba? Napisz program (plik źródłowy: *Bin2Float.java*) wyznaczający tę wartość.

> Liczby typu `int` i `float` zajmują w pamięci 32 bity. Znajdź liczbę całkowitą posiadającą podaną reprezentację binarną i skorzystaj z metody `intBitsToFloat()` z klasy `Float`.

Zadanie 9.4.

Ciąg bitów 11111000001111000011100011001O reprezentuje pewną liczbę zmiennoprzecinkową podwójnej precyzji (zera nieznaczące pominięto). Jaką wartość dziesiętną ma ta liczba? Napisz program (plik źródłowy: *Bin2Double.java*) wyznaczający tę wartość.

> Liczby typu `long` i `double` zajmują w pamięci 64 bity.

Zadanie 9.5.

Napisz binarne reprezentacje symboli specjalnych ($\pm\infty$, *NaN*) dla liczb zmiennoprzecinkowych pojedynczej precyzji. Utwórz aplikację (plik źródłowy: *TestBinFloat.java*), która odkoduje liczbę binarną podaną jako parametr wywołania i wyświetli odpowiadającą jej wartość typu `float`. Wykorzystaj tę aplikację do sprawdzenia poprawności swoich odpowiedzi.

> Reprezentacja binarna liczby zmiennoprzecinkowej pojedynczej precyzji składa się z bitu znaku, *8* bitów cechy i *23* bitów mantysy (razem *32* bity). Cecha symbolu specjalnego ma wszystkie bity równe *1*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 2.

Drugi krok — operacje wejścia-wyjścia i instrukcje sterujące w Javie

10. Wyświetlanie sformatowanych wyników w konsoli. Stałe i metody z klasy Math

Do formatowania wyświetlanych wyników w konsoli możemy użyć metody `printf()` na obiekcie `out` klasy `System`:

```
System.out.printf(łańcuch_formatujący, lista_argumentów);
```

Pierwszy parametr (*łańcuch_formatujący*) jest łańcuchem znaków (obiektem klasy `String`) zawierającym znaczniki formatujące w postaci:

```
%[indeks_argumentu$][flaga][szerokość][.precyzja]znak_konwersji
```

Drugi parametr (*lista_argumentów*) jest listą zmiennych lub stałych oddzielonych przecinkami — wartości tych argumentów będą podstawiane w miejsce znaczników formatujących w łańcuchu formatującym zgodnie z kolejnością występowania lub w miejscu określonym przez opcjonalny *indeks_argumentu$*. Pozostałe opcjonalne parametry określają:

- *flaga* — znak określający dodatkowe cechy sformatowanego wyniku, np. + oznacza, że przed liczbą zawsze wyświetlany jest znak + lub -.

Ebookpoint.pl kopia dla: pawher@o2.pl

- *szerokość* — określa szerokość pola, czyli liczbę znaków przeznaczonych do wyświetlenia wartości argumentu (jeśli wartość nie mieści się w tym polu, to jego rozmiar zostanie automatycznie poszerzony).
- *precyzja* — określa liczbę miejsc po przecinku (domyślnie 6) dla wyświetlanych liczb zmiennoprzecinkowych.

Znacznik formatujący rozpoczyna się znakiem `%` i kończy (obowiązkowym) znakiem (*znak_konwersji*) określającym typ formatowanych danych, np.: `c` — znak, `d` — liczba całkowita dziesiętna, `f` — liczba zmiennoprzecinkowa, `s` — łańcuch znaków. Więcej szczegółów znajdziemy w dokumentacji klasy `Formater`: *http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html*.

Klasa `Math` zawiera zestaw funkcji matematycznych i dwie stałe (pola klasy) — przybliżenie liczby π i podstawy logarytmu naturalnego *e*. Wszystkie funkcje zostały zdefiniowane jako metody statyczne, więc nie tworzymy obiektów klasy `Math`, a jedynie wywołujemy zdefiniowane metody, np. `Math.sqrt(2)` zwróci wartość (przybliżoną) pierwiastka kwadratowego z liczby 2. Wykaz funkcji znajdziemy w dokumentacji: *http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html*.

Zadanie 10.1.

Przedstaw ułamek $\frac{4}{7}$ z dokładnością do 5 miejsc po przecinku.

Dzieląc 4 przez 7, otrzymasz ułamek dziesiętny, o ile co najmniej jedną z tych liczb zamienisz na liczbę zmiennoprzecinkową, np. stosując tzw. rzutowanie `double x = (double) 4`. Do sformatowania wyniku możesz użyć metody `System.out.printf()`.

Zadanie 10.2.

Wyświetl w konsoli z dokładnością do 10 miejsc po przecinku następujące liczby niewymierne: *e*, π i φ (liczba Fibonacciego). Liczby poprzedź komentarzem słownym i ustaw w kolumnie wyrównanej do prawej strony.

Liczby *e* i π są zdefiniowane w klasie `Math` jako stałe, natomiast $\varphi = \frac{1+\sqrt{5}}{2}$.

Zadanie 10.3.

Napisz aplikację, która wyświetli w konsoli w trzech kolumnach liczby naturalne 2, 3, 5, 7, 11, 13 i 17, pierwiastki kwadratowe i pierwiastki sześcienne z tych liczb. Wartości pierwiastków wyświetlaj z dokładnością do 8 miejsc po przecinku, w kolumnach o szerokości 15 znaków.

Liczby 2, 3, 5, 7, 11, 13 i 17 umieść w tablicy. Zastosuj pętlę typu `for each` do przeglądania tablicy, obliczeń i wyświetlania pierwiastków.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 10.4.

Napisz aplikację, która wyświetli w konsoli pierwiastki arytmetyczne od stopnia 2. do 10. z liczby 5 z dokładnością do 6 miejsc po przecinku.

> Zgodnie z definicją $\sqrt[n]{x} = x^{\frac{1}{n}}$, więc do obliczania pierwiastków można wykorzystać metodę `Math.pow(`*podstawa*`, `*wykładnik*`)`.

Zadanie 10.5.

Wyświetl w konsoli kody ósemkowe, dziesiątkowe i szesnastkowe wielkich liter alfabetu łacińskiego. W pierwszym wierszu umieść opisy poszczególnych kolumn: *Znak*, *OCT*, *DEC* i *HEX*.

> Wykorzystaj specyfikatory `%o`, `%d` i `%x` w łańcuchu formatującym.

Zadanie 10.6.

Wyświetl w konsoli miarę kąta o rozwartości 1 radiana w stopniach (z maksymalną możliwą precyzją), w stopniach i minutach kątowych oraz w stopniach, minutach i sekundach kątowych.

> W klasie `Math` znajdziesz metody zamieniające radiany na stopnie i stopnie na radiany. Pozostaje opracować samodzielnie zamianę części dziesiętnych stopnia na minuty (1° = 60') i sekundy kątowe (1' = 60").

Zadanie 10.7.

Wyświetl w konsoli miary kątów 1°, 1' i 1" w radianach z dokładnością do 15 miejsc po przecinku.

Zadanie 10.8.

Oblicz kąty ostre w trójkącie egipskim (trójkącie prostokątnym o proporcji boków 3:4:5). Wyniki podaj:

a) w radianach, z dokładnością do 4 miejsc po przecinku,

b) w stopniach, z dokładnością do 1 miejsca po przecinku,

c) w stopniach i minutach kątowych, z dokładnością do 1',

d) w stopniach, minutach i sekundach kątowych z dokładnością do 1".

> W klasie `Math` znajdziesz funkcje trygonometryczne (argumenty w radianach) oraz funkcje do nich odwrotne (wartości w radianach).

Ebookpoint.pl kopia dla: pawher@o2.pl

11. Wczytywanie danych — klasa Scanner

Pobieranie danych ze standardowego strumienia wejściowego `System.in` możemy zrealizować za pomocą obiektu klasy `Scanner`:

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

Klasa `Scanner` oferuje m.in. metody testujące, czy w strumieniu znajduje się kolejny token określonego typu, oraz metody odczytujące jego wartość ze strumienia:

- `hasNextLine()` — zwraca wartość `true`, gdy w strumieniu znajduje się wiersz danych, w przeciwnym wypadku zwraca wartość `false`;
- `nextLine()` — odczytuje jeden wiersz danych;
- `hasNext()` — zwraca wartość `true`, gdy w strumieniu znajduje się następny token (np. słowo ograniczone spacjami);
- `next()` — odczytuje ze strumienia jeden token (jedno słowo);
- `hasNextInt()` — zwraca wartość `true`, gdy kolejny token w strumieniu reprezentuje liczbę całkowitą;
- `nextInt()` — odczytuje ze strumienia kolejny token, interpretując go jako liczbę całkowitą.

Podobne metody zdefiniowano dla pozostałych prostych typów danych. Szczegółowe informacje znajdziemy w dokumentacji pod adresem *http://docs.oracle.com/javase/1.5.0/docs/api/java/util/Scanner.html*.

Zadanie 11.1.

Napisz program zamieniający temperaturę podaną w stopniach Celsjusza na temperaturę wyrażoną w stopniach Fahrenheita. Dane wejściowe wprowadzamy z klawiatury w postaci liczby dziesiętnej; wynik należy obliczyć i wyświetlić z dokładnością do 0,1 stopnia.

Wzór do zamiany temperatury ze skali Celsjusza na skalę Fahrenheita: °F = °C·1,8+32.

Zadanie 11.2.

Napisz program zamieniający temperaturę podaną w stopniach Fahrenheita na temperaturę wyrażoną w stopniach Celsjusza. Dane wejściowe wprowadzamy z klawiatury w postaci liczby całkowitej, wynik należy obliczyć i wyświetlić z dokładnością do 0,1 stopnia.

Przekształć wzór podany w zadaniu 11.1.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 11.3.

Napisz program obliczający długość przeciwprostokątnej w trójkącie prostokątnym, mając dane:

a) długości przyprostokątnych,

b) długość przyprostokątnej i miarę kąta ostrego (podaną w stopniach) leżącego naprzeciw tej przyprostokątnej.

Dane użytkownik wprowadza z klawiatury. Wyniki należy wyświetlić z dokładnością do *0,001*.

Zadanie 11.4.

Napisz program, który odczyta z konsoli dwie liczby zapisane w naturalnym kodzie binarnym, obliczy ich sumę i wyświetli wynik w postaci binarnej. Zaproponuj ograniczenie liczby cyfr binarnych stosownie do przyjętej metody obliczeń i użytych zmiennych.

Odczytaj ciągi cyfr binarnych i zamień je na liczby całkowite, dodaj uzyskane rezultaty i przedstaw wynik w postaci binarnej. Później wykonamy te obliczenia na ciągach cyfr, bez konwersji na liczby.

Zadanie 11.5.

Napisz program, który zamieni ułamek zwykły na procent. Ułamek zwykły wprowadzamy z klawiatury w postaci łańcucha znaków złożonego z dwóch liczb całkowitych (licznika i mianownika) oddzielonych znakiem / (bez odstępów). Wynik należy wyświetlić z dokładnością do 0,1%.

Zadanie 11.6.

Łańcuch znaków zawiera oddzielone odstępami imię i nazwisko pracownika, liczbę przepracowanych godzin i stawkę godzinową. Napisz program obliczający na tej podstawie wynagrodzenie należne pracownikowi.

Wykorzystaj metody klasy `Scanner` do przeczytania danych z łańcucha znaków.

Zadanie 11.7.

Plik tekstowy *dane.txt* zawiera wiersz tekstu, a w nim oddzielone odstępami imię i nazwisko pracownika, liczbę przepracowanych godzin i stawkę godzinową. Napisz program obliczający na tej podstawie wynagrodzenie należne pracownikowi.

Wykorzystaj metody klasy `Scanner` do przeczytania danych z pliku tekstowego.

Ebookpoint.pl kopia dla: pawher@o2.pl

12. Operacje na tekstach — klasy StringBuffer i StringBuilder

Składanie łańcucha znaków z krótszych łańcuchów możemy zrealizować przy użyciu obiektu klasy `StringBuffer` lub `StringBuilder` (od wersji JDK 5.0).

Obie klasy mają ten sam interfejs. Podstawowe metody to: `append()` (dodawanie znaków na końcu łańcucha — dane różnych typów są konwertowane na typ `String` i dołączane do budowanego obiektu), `delete()` (usuwanie znaków z łańcucha), `insert()` (wstawianie znaków do łańcucha), `replace()` (zastępowanie znaków w łańcuchu). W budowanym obiekcie mamy dostęp do odczytu pojedynczych znaków (metoda `charAt()`) i podłańcuchów (metoda `substring()`) oraz możliwość zmiany pojedynczego znaku (`setCharAt()`).

Więcej informacji można znaleźć w dokumentacji: *http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/StringBuffer.html*.

Zadanie 12.1.

Wykresem funkcji kwadratowej $f(x) = ax^2 + bx + c, \quad a \neq 0$ jest parabola o wierzchołku $\left(\frac{-b}{2a}, \frac{-\Delta}{4a}\right)$. Napisz program, który obliczy i wyświetli w konsoli współrzędne wierzchołka paraboli. Zakładamy, że użytkownik będzie podawał poprawne współczynniki trójmianu.

> Wykorzystaj obiekt klasy `StringBuffer` lub `StringBuilder` i metodę `append()` do zbudowania łańcucha znaków przedstawiającego współrzędne wierzchołka. Metoda `append()` dodaje na końcu łańcucha znaki, łańcuchy znaków, wartości wbudowanych typów danych zamienione na łańcuch znaków lub obiekty zamienione na łańcuch znaków.

Zadanie 12.2.

Rozwiąż zadanie 12.1, stosując obiekt i metodę `insert()` klasy `StringBuffer` lub `StringBuilder`.

> Zacznij od zbudowania łańcucha znaków `"(, )"`, do którego następnie wstawisz obliczone wartości współrzędnych wierzchołka paraboli. Zwróć uwagę na kolejność wstawiania tych wartości do początkowego łańcucha.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 12.3.

Rozwiąż zadanie 12.1, stosując obiekt i metodę `replace()` klasy `StringBuffer` lub `StringBuilder`.

Zacznij od zbudowania łańcucha znaków `"(#1, #2)"`. Podłańcuchy `"#1"` i `"#2"` (o długości 2 znaków) wskazują miejsca, w których zastąpisz te podłańcuchy obliczonymi wartościami współrzędnych wierzchołka paraboli. Aby znaleźć miejsca wstawiania obliczonych wartości, zastosuj metodę `indexOf()`.

Zadanie 12.4.

Napisz program, który obliczy różnicę liczby całkowitej dodatniej wprowadzonej z konsoli i liczby zapisanej tymi samymi cyframi w odwrotnej kolejności. Wynik wyświetl w postaci całego wyrażenia, np.: `452-254 = 198`.

W klasie `StringBuffer` lub `StringBuilder` znajdziesz metodę `reverse()` odwracającą kolejność znaków w łańcuchu oraz metodę `toString()` konwertującą budowany łańcuch na obiekt klasy `String`.

Zadanie 12.5.

Symbol `[a, b]` w geometrii analitycznej oznacza wektor o współrzędnych `a` i `b` (w przestrzeni dwuwymiarowej, czyli na płaszczyźnie). Napisz program obliczający długość wektora `[a, b]` wprowadzonego przez użytkownika z klawiatury w postaci łańcucha znaków, np. `[2.5, -4]`. Zakładamy, że użytkownik będzie podawał poprawną postać danych (nawiasy prostokątne, przecinek i liczby dziesiętne w notacji z kropką).

Z łańcucha znaków należy wydobyć podłańcuchy reprezentujące liczby `a` i `b` (współrzędne wektora), zamienić je na liczby i obliczyć długość wektora ze wzoru $\left|\vec{u}\right| = \sqrt{a^2 + b^2}$.

Zadanie 12.6.

Napisz program obliczający długość wektora `[a, b, c]` (w przestrzeni trójwymiarowej) wprowadzonego przez użytkownika z klawiatury w postaci łańcucha znaków. Zakładamy, że użytkownik będzie podawał poprawną postać danych (nawiasy prostokątne, dwa przecinki oddzielające współrzędne i trzy liczby dziesiętne w notacji z kropką).

Długość wektora obliczymy ze wzoru $\left|\vec{u}\right| = \sqrt{a^2 + b^2 + c^2}$.

Ebookpoint.pl kopia dla: pawher@o2.pl

13. Instrukcje warunkowe i instrukcja selekcji

Instrukcje warunkowe stosujemy w sytuacji, gdy dalszy przebieg programu zależy od spełnienia pewnego warunku (wyrażenia logicznego o wyniku typu `boolean`).

Instrukcja warunkowa ma postać

```
if (wyrażenie) instrukcja;
```

lub

```
if (wyrażenie) instrukcja1; else instrukcja2;
```

Najpierw obliczana jest wartość wyrażenia logicznego. Jeżeli wyrażenie ma wartość `true`, to wykonywana jest `instrukcja` (lub `instrukcja1` w drugiej postaci instrukcji warunkowej). Jeżeli wyrażenie ma wartość `false`, to wykonanie `instrukcji` w pierwszym przypadku jest pomijane, a w drugim przypadku wykonywana jest `instrukcja2`.

Instrukcje mogą być instrukcjami prostymi, złożonymi (ciąg instrukcji ujętych w nawiasy klamrowe {}, tzw. blok) lub instrukcjami warunkowymi (tzw. zagnieżdżanie instrukcji warunkowych).

Instrukcja selekcji pozwala na wybór jednego z kilku wariantów. Składnia instrukcji jest następująca:

```
switch (wyrażenie) {
    case wartość1: instrukcja1;
        break;
    case wartość2: instrukcja2;
        break;
    case wartość3: instrukcja3;
        break;
    // itd.
....default: instrukcja;
        break;
}
```

Wyrażenie przyjmuje wartości całkowite. Instrukcja (`switch`) porównuje wartość wyrażenia z wartościami *wartość1*, *wartość2* itd. Jeśli jedna z nich jest równa wartości obliczonego wyrażenia, to wykonywana jest odpowiednia instrukcja z ciała instrukcji `switch`. Instrukcja `break` powoduje natychmiastowe zakończenie instrukcji `switch`. Pominięcie instrukcji `break` jest dopuszczalne, ale spowoduje wykonywanie kolejnych instrukcji występujących po wybranej instrukcji (do końca instrukcji selekcji lub wystąpienia `break`). Jeśli wyrażenie nie przyjmuje żadnej z podanych wartości, to wykonywana jest instrukcja domyślna (`default`). Instrukcja domyślna jest opcjonalna i możemy ją pominąć.

Instrukcje selekcji można zagnieżdżać.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.1.

Napisz program, który zapyta użytkownika o imię, a następnie wykorzystując podane imię, wyświetli w konsoli napis w postaci: `"Jan jest mężczyzną."` lub `"Anna jest kobietą."`.

> Wykorzystaj fakt, że typowe polskie imiona żeńskie są zakończone na literę `a`.

Zadanie 13.2.

Napisz program obliczający pole powierzchni i obwód kwadratu. Długość boku kwadratu użytkownik wprowadzi z klawiatury. Program powinien sprawdzić, czy wprowadzona długość boku jest poprawna (dodatnia).

Zadanie 13.3.

Użytkownik wprowadza z klawiatury dwie liczby rzeczywiste `a` i `b`. Napisz program, który wyświetli (zapisany w sposób symboliczny) zbiór złożony z tych liczb oraz wszystkich liczb zawartych między nimi.

> Jeśli liczby są równe, to otrzymamy zbiór jednoelementowy {a}, w przeciwnym razie szukanym zbiorem będzie przedział domknięty.

Zadanie 13.4.

Użytkownik wprowadza z klawiatury współczynniki funkcji kwadratowej

$$f(x) = ax^2 + bx + c$$

Napisz program, który wyznaczy zbiór wartości tej funkcji lub wyświetli komunikat, że podane współczynniki nie są poprawne (przypadek $a = 0$).

> Do zapisania zbioru wartości funkcji, który jest przedziałem nieograniczonym, będzie potrzebny symbol ∞. W konsoli możemy ten symbol „złożyć" z dwóch znaków `oo`, co w połączeniu ze znakiem + lub – będzie wystarczająco czytelne (`-oo` lub `+oo`).

Zadanie 13.5.

Użytkownik wprowadza z klawiatury współczynniki funkcji kwadratowej

$$f(x) = ax^2 + bx + c$$

Napisz program, który wyznaczy przedziały monotoniczności i ekstremum tej funkcji.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.6.

Napisz program rozwiązujący równanie liniowe $ax + b = 0$. Współczynniki równania użytkownik wprowadzi z klawiatury. Rozważ wszystkie możliwe przypadki dla współczynników równania.

Zadanie 13.7.

Użytkownik wprowadza z klawiatury trzy liczby `a`, `b` i `c`. Napisz program, który wyświetli najmniejszą (lub największą) z tych liczb. Nie wprowadzaj dodatkowych zmiennych.

Zadanie 13.8.

Użytkownik wprowadza z klawiatury trzy liczby `a`, `b` i `c`. Napisz program, który wyświetli te liczby w sposób uporządkowany, w kolejności od najmniejszej do największej (lub od największej do najmniejszej). Nie wprowadzaj dodatkowych zmiennych.

Zadanie 13.9.

Stosując instrukcję selekcji, napisz program, który liczbę naturalną jednocyfrową zapisze słownie.

Zadanie 13.10.

Napisz program, który podaną liczbę naturalną mniejszą od 100 zapisze słowami.

Zadanie 13.11.

Napisz program, który podaną liczbę naturalną mniejszą od 1000 zapisze słowami.

Zadanie 13.12.

Napisz program, który sprawdzi poprawność daty z obecnego wieku, zapisanej w postaci `dd.mm.rrrr`, i wyświetli odpowiedni komunikat.

Zadanie 13.13.

Napisz program, który datę podaną w postaci `dd.mm.rrrr` zapisze w formacie `rrrr-mm-dd`. Przyjmij, że wprowadzana data jest poprawna (np. sprawdzona programem z zadania 13.12).

Zadanie 13.14.

Napisz program, który datę podaną w formacie `dd.mm.rrrr` zapisze w postaci tradycyjnej:

a) z miesiącem zapisanym słownie, np. 15 marca 2012;

b) z miesiącem zapisanym w systemie rzymskim, np. 15 III 2012.

Przyjmij, że wprowadzona przez użytkownika data jest poprawna.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.15.

Napisz program, który na podstawie daty podanej w formacie `dd.mm.rrrr` określi, jaki to dzień tygodnia. Przyjmij, że wprowadzona przez użytkownika data jest poprawna.

> Poszukaj informacji na temat wzoru Zellera.

14. Instrukcja pętli typu do-while

Pętla ze sprawdzaniem warunku na końcu ma postać:

```
do
    instrukcja;
while(warunek);
```

Najpierw jest wykonywana *instrukcja*, potem obliczana jest wartość wyrażenia logicznego (*warunek*). Jeśli warunek ma wartość `true`, to powracamy do wykonywania instrukcji, i cykl się powtarza.

Cechą charakterystyczną tej pętli jest to, że instrukcja wykona się co najmniej raz (`do` *instrukcja*`; while(false);`). Należy zwrócić uwagę, aby *instrukcja* miała wpływ na badany warunek, ponieważ możemy uzyskać pętlę nieskończoną (`do` *instrukcja*`; while(true);`).

Pętlę nieskończoną, utworzoną świadomie, możemy przerwać, używając w ciele pętli instrukcji warunkowej i instrukcji `break`.

Zadanie 14.1.

Użytkownik wprowadza z klawiatury serię liczb różnych od zera, zero natomiast jest sygnałem zakończenia wprowadzania danych. Napisz program, który obliczy sumę tych liczb.

Zadanie 14.2.

Podczas wprowadzania danych niejednokrotnie potrzebujemy liczb spełniających określone warunki, np. dodatnich. Napisz program, który przyjmie z konsoli wyłącznie wartość dodatnią zmiennej. Jeśli użytkownik poda liczbę ujemną lub zero, to program powinien ponowić żądanie podania właściwej wartości.

> Wczytywanie wartości zmiennej umieść w pętli ze sprawdzaniem warunku na końcu.

Zadanie 14.3.

Napisz program, który wyświetli w konsoli wielokrotności liczby 7 mniejsze od 100.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 14.4.

Napisz program obliczający pole powierzchni koła. Promień koła użytkownik wprowadza z klawiatury. Program powinien zasygnalizować błędne dane (liczbę ujemną lub zero) i ponownie zapytać o długość promienia.

Zadanie 14.5.

Napisz program obliczający pole powierzchni pierścienia kołowego o promieniu zewnętrznym *R* i wewnętrznym *r*. Długości promieni użytkownik wprowadza z klawiatury. Program powinien zasygnalizować błędne dane i ponownie zapytać o potrzebną wartość.

Zadanie 14.6.

Napisz program wyświetlający w konsoli ciąg liczb Fibonacciego mniejszych od *1000*.

> Ciąg liczb Fibonacciego określamy w następujący sposób: *F(1) = 1*, *F(2) = 1*, *F(n) = F(n–1)+F(n–2)* dla *n =3*, *4*, *5*.... Pomimo rekurencyjnego charakteru wzoru możemy zrealizować obliczenia w sposób iteracyjny, używając trzech zmiennych i odpowiednio przestawiając w nich wartości trzech kolejnych liczb ciągu Fibonacciego.

Zadanie 14.7.

Napisz program obliczający, ile jest liczb w ciągu Fibonacciego mniejszych od podanej przez użytkownika liczby całkowitej *n*.

Zadanie 14.8.

Napisz program obliczający pole powierzchni trójkąta równobocznego i umożliwiający wielokrotne powtarzanie obliczeń. Jako znak zakończenia obliczeń przyjmij podanie długości boku równej 0. Dla wartości ujemnych wyświetl stosowny komunikat i ponów pytanie o długość boku.

Zadanie 14.9.

Napisz program umożliwiający wielokrotne wykonywanie obliczeń (np. długości okręgu). Po wykonaniu obliczeń program powinien wyświetlić pytanie `Czy obliczamy dalej (t/n)?`. W pytaniu zawarta jest sugestia sposobu udzielenia odpowiedzi: `t` — tak, `n` — nie (inne odpowiedzi powinny być ignorowane).

15. Instrukcja pętli typu while

Pętla ze sprawdzaniem warunku na początku ma postać:

```
while(warunek) instrukcja;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Najpierw obliczana jest wartość wyrażenia logicznego *warunek*. Jeżeli *warunek* ma wartość `true`, to wykonywana jest *instrukcja* i następuje powrót do badania warunku (cykl się powtarza). Jeżeli *warunek* przyjmie wartość `false`, to działanie pętli zostanie zakończone.

Cechą charakterystyczną tej pętli jest to, że instrukcja może nie wykonać się ani razu (`while(false) instrukcja;`). Należy zwrócić uwagę, aby *instrukcja* miała wpływ na badany warunek, gdyż możemy uzyskać pętlę nieskończoną (`while(true) instrukcja;`).

Ponieważ składnia pętli dopuszcza użycie instrukcji pustej, powstaje możliwość przypadkowego utworzenia pętli nieskończonej w postaci `while(true); instrukcja;` (wstawienie znaku średnika pomiędzy warunek i instrukcję).

Zadanie 15.1.

Użytkownik wprowadza z klawiatury dwie liczby naturalne. Nie korzystając z operatora dodawania (+), oblicz sumę tych liczb. Napisz program obliczający sumę tych liczb, mając do dyspozycji operatory inkrementacji (++) i dekrementacji (--).


Wskazówka

Wyobraź sobie dwa stosy patyczków. Dodawanie realizujesz, przekładając po jednym patyczku z jednego stosu na drugi. Wykorzystaj ten pomysł do rozwiązania zadania.

Zadanie 15.2.

Użytkownik wprowadza z klawiatury dwie liczby całkowite. Napisz program obliczający sumę tych liczb, korzystając z operatorów inkrementacji (++) i dekrementacji (--).

Zadanie 15.3.

Napisz program obliczający iloczyn liczb całkowitych, nie korzystając z operatora *. Do dyspozycji masz operatory + i --. Zadanie rozwiąż:

a) dla pary liczb całkowitych dodatnich,

b) dla pary dowolnych liczb całkowitych.


Wskazówka

W zadaniu *b* rozważ wszystkie możliwe przypadki i sprowadź je ewentualnie do przypadku z zadania *a*.

Zadanie 15.4.

Korzystając z operatora odejmowania (-), napisz program obliczający iloraz całkowity pary liczb całkowitych.

Zadanie 15.5.

Korzystając z operatora odejmowania (-), napisz program obliczający resztę z dzielenia pary liczb całkowitych.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 15.6.

Napisz program obliczający *całkowity pierwiastek kwadratowy* z podanej liczby naturalnej. Nie korzystaj przy tym z funkcji bibliotecznych oraz innych metod przybliżonego obliczania pierwiastka kwadratowego i zaokrąglenia otrzymanego wyniku zmiennoprzecinkowego do liczby całkowitej.


Wskazówka

Całkowity pierwiastek kwadratowy z liczby *n* jest największą liczbą naturalną *p* spełniającą nierówność $p^2 \leq n$.

Zadanie 15.7.

Napisz program wyznaczający najmniejszą wspólną wielokrotność (NWW) dla pary liczb całkowitych dodatnich.


Wskazówka

Obliczaj wielokrotności jednej liczby i sprawdzaj, czy obliczona wielokrotność jest wielokrotnością drugiej liczby (czy dzieli się bez reszty przez drugą liczbę).

Zadanie 15.8.

Korzystając z algorytmu Euklidesa, napisz program wyznaczający największy wspólny dzielnik (NWD) pary liczb całkowitych dodatnich.

Zadanie 15.9.

Mamy dwie różne liczby całkowite dodatnie. Bierzemy mniejszą z nich i obliczamy jej wielokrotność do chwili, gdy wielokrotność osiągnie lub przekroczy wartość drugiej liczby. W przypadku równości mamy już gotowy wynik. Jeśli jednak wielokrotność pierwszej liczby przekroczyła wartość drugiej liczby, to zaczynamy liczyć wielokrotności drugiej liczby i porównywać wyniki z obliczoną wielokrotnością pierwszej liczby. Naprzemienne liczenie kolejnych wielokrotności zakończymy w chwili, gdy obie wielokrotności będą równe. Uzasadnij, że to postępowanie się zakończy. Napisz na tej podstawie program wyznaczający najmniejszą wspólną wielokrotność (NWW) pary liczb całkowitych dodatnich.

16. Instrukcja pętli typu for

Instrukcja pętli typu `for` ma składnię:

```
for(instrukcja_inicjująca; wyrażenie_logiczne; instrukcja_modyfikująca) instrukcja;
```

Na początku wykonywana jest (tylko raz) *instrukcja_inicjująca*. Następnie obliczana jest wartość wyrażenia logicznego (*wyrażenie_logiczne*). Jeżeli wyrażenie logiczne ma wartość `true`, to wykonywana jest *instrukcja*, w przeciwnym razie (gdy wyrażenie logiczne ma wartość `false`) pętla zostaje zakończona. Po wykonaniu instrukcji

Ebookpoint.pl kopia dla: pawher@o2.pl

(*instrukcja*) wykonywana jest *instrukcja_modyfikująca* i następuje powrót do obliczania wartości wyrażenia logicznego (cykl jest kontynuowany).

Instrukcja `for` często przyjmuje postać (`n` jest ustaloną liczbą naturalną):

```
for(int i = 0; i < n; ++i) instrukcja; // odliczanie w górę
```

lub:

```
for(int i = n; i > 0; --i) instrukcja; // odliczanie w dół
```

W obu przypadkach *instrukcja* zostanie wykonana `n` razy, przy czym w pierwszym przypadku zmienna `i` (zwana często zmienną sterującą lub licznikiem) przyjmie kolejno wartości *0*, *1*, *2* … *n-1*, a w drugim przypadku *n*, *n–1*, … *1*. W każdym cyklu *instrukcja_modyfikująca* zmienia wartość zmiennej `i` (`++i` — *inkrementacja*, czyli zwiększenie wartości zmiennej `i` o `1`, lub `--i` — *dekrementacja*, czyli zmniejszenie zmiennej `i` o `1`).

Pętla `for` nie ogranicza się do przedstawionych wyżej zastosowań. W uzasadnionych sytuacjach można poszczególne instrukcje pomijać. W skrajnym przypadku uzyskamy nieskończoną pętlę w postaci `for(;;);`, wykonującą *instrukcję pustą*.

Zadanie 16.1.

Napisz program, który nie wykonując mnożenia, obliczy kwadrat liczby naturalnej.

> Wykorzystaj fakt, że suma kolejnych liczb nieparzystych jest kwadratem liczby naturalnej: $1 = 1^2$, $1 + 3 = 4 = 2^2$, $1 + 3 + 5 = 9 = 3^2$ itd.

Zadanie 16.2.

Napisz program, który sprawdzi, czy wprowadzone z klawiatury słowo jest *palindromem*, czyli brzmi tak samo czytane od strony lewej do prawej i od prawej do lewej. Przykładem palindromu jest słowo *kajak* lub imię *Anna* (bez rozróżniania wielkości liter).

> Możesz porównywać podany tekst z tekstem zapisanym tymi samymi znakami w odwrotnej kolejności lub porównywać pierwszy znak z ostatnim, drugi z przedostatnim itd.

Zadanie 16.3.

Napisz program, który sprawdzi, czy wprowadzone z klawiatury zdanie jest palindromem, czyli brzmi tak samo czytane od strony lewej do prawej i od prawej do lewej. Przykładem zdania-palindromu jest *Kobyła ma mały bok* (bez rozróżniania wielkości liter i uwzględniania odstępów między słowami).

Zadanie 16.4.

Napisz program wyświetlający na ekranie tabliczkę mnożenia.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 16.5.

Napisz program wyświetlający na ekranie tabliczkę dodawania i tabliczkę mnożenia w piątkowym układzie pozycyjnym.

Zadanie 16.6.

Napisz program obliczający sumę *n* początkowych wyrazów ciągu harmonicznego. Liczbę *n* użytkownik wprowadza z klawiatury.

Zadanie 16.7.

Napisz program obliczający silnię liczby naturalnej *n*. Liczbę *n* użytkownik wprowadza z klawiatury.

Zadanie 16.8.

Napisz program umożliwiający wielokrotne obliczanie potęg o wykładniku naturalnym *n* i podstawie rzeczywistej *a*. Nie korzystaj z funkcji klasy `Math`. Podanie wykładnika *–1* powinno przerwać działanie programu.

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 3.

Trzeci krok — budujemy własne metody i klasy

17. Obsługa wyjątków

Wyjątek (ang. *exception*) — nietypowa sytuacja pojawiająca się podczas działania programu. Każdy wyjątek w Javie jest obiektem określonego typu, przekazywanym do metody, której działanie doprowadziło do pojawienia się błędu. Metoda może przechwycić wyjątek i wykonać instrukcje zapobiegające przerwaniu działania programu.

Konstrukcja obsługi wyjątków przedstawia się następująco:

```
try { instrukcja }
catch (typ_wyjątku_1 e) { instrukcja_obsługi_wyjątku_1 }
catch (typ_wyjątku_2 e) { instrukcja_obsługi_wyjątku_2 }
...
catch (typ_wyjątku_n e) { instrukcja_obsługi_wyjątku_n }
[finally { instrukcja }]
```

Opcjonalny blok `finally` jest zawsze wykonywany przed powrotem do instrukcji występującej za konstrukcją `try-catch`.

W kodzie programu możemy generować wyjątki, stosując instrukcję `throw`.

Zadanie 17.1.

Podczas wprowadzania danych liczbowych użytkownik może świadomie lub przez pomyłkę wprowadzić ciąg znaków niebędący poprawnie zapisaną liczbą. Spowoduje to przerwanie pracy programu. Napisz program umożliwiający wczytywanie z klawiatury liczby zmiennoprzecinkowej typu `double` przy użyciu metody `nextDouble()` z klasy `Scanner` i reagujący poprawnie na popełniony błąd.

Należy przechwycić i obsłużyć wyjątek `InputMismatchException` (pakiet `java.util`), a następnie ponowić proces wczytywania liczby.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 17.2.

Dane liczbowe ze standardowego wejścia możemy wczytywać w postaci łańcucha znaków, a następnie konwertować je na liczby odpowiedniego typu. Napisz program, który w ten sposób wczyta z klawiatury liczbę zmiennoprzecinkową i zareaguje poprawnie na popełnione błędy.

> Podczas konwersji łańcucha na liczbę może wystąpić wyjątek `NumberFormatException`. Należy przechwycić i obsłużyć ten wyjątek, a następnie ponowić proces wczytywania danych.

Zadanie 17.3.

Napisz program, który odczyta i obliczy sumę pięciu liczb całkowitych, wprowadzonych ze standardowego wejścia. Pomiń liczby zmiennoprzecinkowe i łańcuchy znaków niebędące liczbami.

> Wykorzystaj wyjątki lub informację, że klasa `Scanner` oferuje metody sprawdzające, jaki kolejny token znajduje się w strumieniu. Na przykład metoda `hasNextInt()` sprawdza, czy kolejny token jest liczbą całkowitą typu `int`.

Zadanie 17.4.

Napisz program obliczający odwrotność liczby całkowitej wprowadzonej z klawiatury.

a) Przechwyć i obsłuż wszystkie wyjątki, jakie mogą pojawić się podczas wczytywania danych i wykonywania obliczeń.

b) Zrealizuj zadanie bez obsługi wyjątków.

Zadanie 17.5.

Napisz program obliczający wynik dzielenia z resztą dwóch liczb całkowitych wprowadzonych z klawiatury. Przechwyć i obsłuż wyjątki, jakie mogą pojawić się podczas wczytywania danych i wykonywania obliczeń.

18. Liczby pseudolosowe i tablice jednowymiarowe — budujemy metody statyczne

Liczby losowe powstają w wyniku działania mechanizmu losującego (rzut kostką do gry, losowanie ponumerowanych kul z urny itp.) — rozkład tych liczb jest nieregularny i trudny do przewidzenia. Otrzymywanie liczb losowych w programach komputerowych jest bardzo skomplikowane, więc często zastępuje się je *liczbami pseudoloso-*

Ebookpoint.pl kopia dla: pawher@o2.pl

wymi (rozkład tych liczb ma pewne regularności, które w wielu zastosowaniach możemy pominąć).

Do generowania liczb pseudolosowych możemy stosować *statyczną metodę* `java.lang.Math.random()`, zwracającą liczbę typu `double` z przedziału $\langle 0, 1)$, lub *obiekt* klasy `java.util.Random` i jedną z dostępnych w klasie metod.

Tablica jednowymiarowa — struktura danych do przechowywania określonej liczby danych jednego typu.

Ogólna definicja tablicy przedstawia się następująco:

```
typ_danych [] nazwa_tablicy = new typ_danych[rozmiar_tablicy];
```

natomiast tablica zainicjowana podanymi wartościami to:

```
typ_danych [] nazwa_tablicy = { wartość_1, wartość_2, ..., wartość_n };
```

W Javie tablica jest obiektem. Rozmiar tablicy zapisany jest w polu `length`. Indeksowanie elementów tablicy rozpoczyna się od `0`. Jeśli podamy indeks spoza zakresu (`0..rozmiar_tablicy-1`), to zostanie wygenerowany wyjątek `java.lang.ArrayIndexOutOfBoundsException`.

Metody statyczne, dostępne w klasie `java.util.Arrays` pozwalają na wykonywanie podstawowych operacji na tablicy — sortowanie, wyszukiwanie binarne, kopiowanie itp.

Zadanie 18.1.

Napisz program, który przeprowadzi symulację 100 rzutów kostką i wyświetli wyniki w konsoli.

Zadanie 18.2.

Napisz program, który przeprowadzi symulację 1000 rzutów kostką i sporządzi zestawienie wyników.


Wskazówka

Nie zapamiętuj wyników kolejnych rzutów — wystarczy, że będziesz na bieżąco zliczał, ile razy wypadł dany wynik. Rolę licznika może odgrywać tablica.

Zadanie 18.3.

Dwukrotnie rzucamy kostką do gry i zapisujemy sumę wyrzuconych oczek. Napisz program, który przeprowadzi symulację 3000 powtórzeń doświadczenia i sporządzi zestawienie wyników.

Zadanie 18.4.

Utwórz funkcję `rand()` z dwoma parametrami `a` i `b`, losującą liczbę rzeczywistą należącą do przedziału $\langle a, b \rangle$, gdzie `a` i `b` są liczbami całkowitymi i `a < b`. Wynik losowania powinien być podany z precyzją do *0,1*. Napisz program demonstrujący działanie tej funkcji.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 18.5.

Napisz program losujący *500* liczb rzeczywistych z przedziału $\langle 0, 5)$. Przedstaw rozkład wylosowanych liczb w przedziałach o długości *1*.

Zadanie 18.6.

Utwórz funkcję (metodę statyczną) `lotto()` losującą *6* różnych liczb z *49* i zwracającą wynik w postaci tablicy liczb całkowitych. Napisz program demonstrujący działanie tej metody.

Zadanie 18.7.

Utwórz funkcję (metodę statyczną) `lotto()` z dwoma parametrami `m` i `n`, losującą `m` różnych liczb spośród liczb od `1` do `n` i zwracającą wynik w postaci tablicy liczb całkowitych. Napisz program demonstrujący działanie tej metody.

Rozwiązując kolejne zadania (18.8 – 18.18), zbudujemy klasę `MyRandomArray`, gromadzącą metody umożliwiające tworzenie jednowymiarowych tablic wypełnionych pseudolosowymi liczbami typu całkowitego `int` lub zmiennoprzecinkowego `double`.

Zadanie 18.8.

Utwórz statyczną metodę `int[] rndArray(int n, int m)` służącą do budowania tablicy liczb całkowitych o podanym wymiarze `n`, wypełnionej pseudolosowymi wartościami z zakresu `0`, `1`, …, `m-1`. Zdefiniuj metody ułatwiające wyświetlanie elementów tablicy oddzielonych odstępem `void arrayPrint(int[] tab)` i `void arrayPrintln(int[] tab)` — nazwy sugerują różnicę w działaniu tych metod. Utwórz również metodę dodającą do wszystkich elementów tablicy określoną wartość `void addToArray(int[] tab, int d)`. Napisz program demonstrujący działanie tych metod.


Uwaga

Tworzone metody skopiuj do pliku *MyRandomArray.java* w bieżącym folderze.

```
import java.util.Random;
public class MyRandomArray {
    /* Tu skopiuj kody metod */
}
```

Będziesz mógł z nich korzystać podczas rozwiązywania innych zadań, np. kod:

```
int[] los = MyRandomArray.rndArray(6, 49);
MyRandomArray.addToArray(los, 1);
MyRandomArray.arrayPrintln(los);
```

stanowi przykład losowania 6 liczb z 49 (podobnie jak w rozwiązaniu zadania 18.6 — w tym przypadku jednak liczby mogą się powtarzać) przy użyciu metod statycznych z klasy `MyRandomArray`.

Zadanie 18.9.

Utwórz statyczną metodę `int[] rndUniqueArray(int n, int m)` służącą do budowania tablicy liczb całkowitych o podanym wymiarze (`n`) i niepowtarzających się elemen-

Ebookpoint.pl kopia dla: pawher@o2.pl

tach, wypełnionej pseudolosowymi wartościami z określonego zakresu 0, 1, …, m-1. Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy MyRandomArray.

Skorzystaj z rozwiązania zadania 18.6.

Zadanie 18.10.

Utwórz statyczne metody int[] rndSortArray(int n, int m) i int[] rndSortUniqueArray(int n, int m), które będą służyć do budowania posortowanych tablic liczb całkowitych o podanym wymiarze (n), wypełnionych pseudolosowymi wartościami z zakresu 0, 1, …, m-1. Druga metoda powinna zwracać tablicę o niepowtarzających się elementach. Napisz program demonstrujący działanie tych metod. Dołącz je do klasy MyRandomArray.

Skorzystaj z rozwiązania zadania 18.9 oraz metody sortującej z klasy Arrays.

Zadanie 18.11.

Utwórz statyczną metodę double[] rndArray(int n, double a), która będzie służyć do budowania tablicy liczb zmiennoprzecinkowych typu double o podanym wymiarze (n), wypełnionej pseudolosowymi wartościami z przedziału $\langle 0, a)$. Zdefiniuj metody ułatwiające wyświetlanie elementów tablicy oddzielonych odstępem void arrayPrint(double[] tab), void arrayPrintln(double[] tab) i void arrayPrintf(String spec, double[] tab) — nazwy sugerują sposób działania tych metod. Ponadto utwórz metodę dodającą do wszystkich elementów tablicy określoną wartość void addToArray(double[] tab, double d). Napisz program demonstrujący działanie tych metod.

Skorzystaj z rozwiązania zadania 18.8.

Metody te mają nazwy takie same jak metody istniejące w klasie MyRandomArray, ale różnią się typami parametrów wywołania. Możemy więc te metody dodawać do klasy MyRandomArray — wykorzystujemy tutaj możliwość *przeciążania metod*.

Zadanie 18.12.

Utwórz statyczną metodę double[] rndUniqueArray(int n, double a), która będzie służyć do budowania tablicy liczb zmiennoprzecinkowych o podanym wymiarze (n) i niepowtarzających się elementach, wypełnionej pseudolosowymi wartościami z przedziału $\langle 0, a)$. Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy MyRandomArray.

Skorzystaj z rozwiązania zadania 18.9.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 18.13.

Utwórz statyczne metody `double[] rndSortArray(int n, double a)` i `double[] rndSortUniqueArray(int n, double a)`, które będą służyć do budowania posortowanych tablic liczb zmiennoprzecinkowych o podanym wymiarze (`n`), wypełnionych pseudolosowymi wartościami z przedziału $\langle 0, a)$. Druga metoda powinna zwracać tablicę o niepowtarzających się elementach. Napisz program demonstrujący działanie tych metod. Dołącz je do klasy `MyRandomArray`.


Wskazówka

Skorzystaj z rozwiązania zadania 18.10.

Zadanie 18.14.

Utwórz statyczną metodę `void roundArray(double[] tab, int prec)`, która będzie zaokrąglać wszystkie liczby w tablicy `tab` do określonej liczby miejsc po przecinku (parametr `prec`). Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy `MyRandomArray`.

Zadanie 18.15.

Utwórz statyczną metodę `double[] rndArray(int n, double a, int prec)`, która będzie służyć do budowania tablicy liczb zmiennoprzecinkowych o podanym rozmiarze (`n`) i wartościach z przedziału $\langle 0, a)$ oraz o określonej liczbie miejsc po przecinku (parametr `prec`). Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy `MyRandomArray`.

Zadanie 18.16.

Utwórz statyczną metodę `double[] rndArray(int n, double a, double step)`, która będzie służyć do budowania tablicy liczb zmiennoprzecinkowych o podanym rozmiarze (`n`) i wartościach z przedziału $\langle 0, a)$ będących wielokrotnością parametru `step`. Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy `MyRandomArray`.

Zadanie 18.17.

Utwórz statyczne metody `void inputArray(int[] tab)` i `void inputArray(double[] tab)`, które będą umożliwiać wprowadzanie danych z klawiatury do tablicy odpowiedniego typu. Napisz program demonstrujący działanie tych metod. Utwórz klasę `MyArray` i umieść w niej te dwie metody oraz wcześniej zdefiniowane metody służące do wyświetlania zawartości tablic jednowymiarowych i do zaokrąglania wartości w tablicy liczb typu `double` (z klasy `MyRandomArray`).

Zadanie 18.18.

Utwórz statyczną metodę konwertującą tablicę liczb całkowitych na tablicę liczb zmiennoprzecinkowych (`double[] intArrayToDouble(int[] tab)`) oraz metodę (`int[] doubleArrayToInt(double[] tab)`) działającą odwrotnie.

Ebookpoint.pl kopia dla: pawher@o2.pl


Wskazówka

Utwórz drugą tablicę o takim samym rozmiarze, z odpowiednim typem danych, i przepisz do niej elementy z pierwszej tablicy, stosując rzutowanie typów.

19. Dokumentacja klasy

JDK zawiera narzędzie `javadoc`, generujące na podstawie *komentarzy dokumentacyjnych* w plikach źródłowych dokumentację w formie plików *HTML*.

W komentarzach można umieszczać *dowolny tekst*, *znaczniki języka HTML* oraz specjalne *znaczniki dokumentacyjne* (zaczynające się od znaku `@`).

Komentarz dokumentacyjny zaczyna się od znaku `/**` i kończy się znakiem `*/`. Komentarze blokowe `/* ... */` i liniowe `// ...` są podczas tworzenia dokumentacji pomijane.

W kolejnych zadaniach poznasz wybrane przykłady z zakresu tworzenia dokumentacji.

Zadanie 19.1.

Przygotuj „ściągawkę" z instrukcją obsługi aplikacji *javadoc.exe* dla używanego środowiska JDK.


Wskazówka

W konsoli wykonaj polecenie `javadoc -help > opis.txt`. W pliku *opis.txt* znajdziesz wszystkie opcje aplikacji *javadoc.exe*.

Zadanie 19.2.

Utwórz plik źródłowy *Z19_2.java*, który będzie zawierał następujący kod:

```
public class Z19_2 {
    private static final String HELLO = "Hello World!";

    static String hello() {
        return HELLO;
    }

    public static void main(String args[]) {
        System.out.println(hello());
    }
}
```

Skompiluj i uruchom przykład. Dokumentację klasy `Z19_2` możesz sporządzić, postępując według schematu:

- otwórz konsolę,
- przejdź do folderu z plikiem *Z19_2.java*,

Ebookpoint.pl kopia dla: pawher@o2.pl

- wykonaj polecenie: `javadoc Z19_2.java` (zapoznaj się z treścią uwagi),
- otwórz plik *index.html* i obejrzyj dokumentację klasy.

Powtórz zadanie, tworząc kilka wersji dokumentacji — zapamiętaj znaczenie użytych opcji:

a) `javadoc -d Z19_2a -nohelp Z19_2.java`

b) `javadoc -d Z19_2b -notree Z19_2.java`

c) `javadoc -d Z19_2c -noindex Z19_2.java`

d) `javadoc -d Z19_2d -nodeprecatedlist Z19_2.java`

e) `javadoc -d Z19_2e -nodeprecatedlist -noindex -notree -nohelp -private Z19_2.java`


Uwaga

Aby uniknąć kolizji pomiędzy plikami dokumentacji (nadpisywanie plików), przyjmij za zasadę tworzenie dokumentacji dla poszczególnych zadań w odrębnych folderach. Nazwy folderów niech zawierają numer zadania. W związku z tym już od tego zadania używaj opcji `-d`, wskazując folder, w którym zostanie zapisana dokumentacja. Zatem zamiast polecenia `javadoc Z19_2.java` zastosuj polecenie `javadoc -d Z19_2 Z19_2.java`. Dokumentacja klasy zostanie zapisana w bieżącym folderze, w podfolderze `Z19_2`.

Zadanie 19.3.

Utwórz plik źródłowy *Z19_3.java*, który będzie zawierał kod z zadania 19.2. Wstaw komentarze w postaci `/** tekst komentarza */` przed kodem klasy, definicją stałej i definicjami metod. W komentarzach zawrzyj zwięzłe opisy przedstawianych elementów. Sporządź dokumentację i przeanalizuj otrzymany dokument. Zastosuj opcje `-d <directory>` i `-private`.

Zadanie 19.4.

Rozwiązując zadania 18.8, 18.11, 18.17 i 18.18, utworzyliśmy kilka statycznych metod ułatwiających pracę z jednowymiarowymi tablicami liczb całkowitych typu `int` i tablicami liczb zmiennoprzecinkowych typu `double`. Metody zostały zgromadzone w klasie `MyArray`. Na tej podstawie utwórz klasę `MyIntArray`, która będzie zawierać metody działające na tablicach liczb całkowitych (możesz zmienić nazwy metod lub dodać dodatkowe metody), i sporządź dokumentację tej klasy (dokumentację zapisz w folderze *MyIntArray*). W dokumentacji umieść informacje o autorze i wersji klasy.


Wskazówka

W komentarzu do klasy wykorzystaj znaczniki `@author` i `@version`. Dokumentację utwórz z opcjami `-author` i `-version`.

Zadanie 19.5.

Na podstawie rozwiązań zadań 18.8, 18.11, 18.17, 18.18 i 19.4 zbuduj klasę `MyDoubleArray`, która będzie zawierać statyczne metody ułatwiające pracę z jednowymiarowy-

Ebookpoint.pl kopia dla: pawher@o2.pl

mi tablicami liczb typu `double`. Sporządź dokumentację klasy i zapisz ją w folderze *MyDoubleArray*.

Zadanie 19.6.

Na podstawie rozwiązania zadania 19.5 zbuduj klasę `MyFloatArray`, która będzie zawierać statyczne metody ułatwiające pracę z jednowymiarowymi tablicami liczb typu `float`. Sporządź dokumentację klasy i zapisz ją w folderze *MyFloatArray*.

Utwórz wspólną dokumentację klas `MyIntArray`, `MyDoubleArray` i `MyFloatArray` w folderze *MyArrays*.

Zadanie 19.7.

W pliku *MyRandomArray.java*, utworzonym podczas rozwiązywania zadań z rozdziału 18., wpisz komentarze dokumentacyjne. Utwórz dokumentację klasy `MyRandomArray` i zapisz tę dokumentację w folderze o tej samej nazwie.

20. Działania na ułamkach — budujemy klasę Fraction

Rozwiązując systematycznie kolejne zadania, będziemy tworzyć własną klasę `Fraction` (ułamki) oraz programy ukazujące możliwości tej klasy. Na bieżąco zbudujemy dokumentację klasy.

Zadanie 20.1.

Zbuduj klasę `Fraction`, która będzie zawierać dwa prywatne pola reprezentujące licznik i mianownik ułamka. W klasie tej umieść konstruktor z dwoma parametrami, który będzie budować ułamek na podstawie dwóch liczb całkowitych (licznika i mianownika), oraz publiczną metodę `toString()`, zwracającą ułamek w postaci łańcucha znaków, np. `"4/13"` (licznik, kreska ułamkowa `/` i mianownik). Napisz program pokazujący działanie konstruktora i zdefiniowanej metody.

Zadanie 20.2.

Dodaj do klasy `Fraction` konstruktor bezparametrowy budujący ułamek odpowiadający liczbie *0* oraz konstruktor z jednym parametrem całkowitym `m` budujący ułamek `m/1`. Napisz program pokazujący działanie tych konstruktorów.

Zadanie 20.3.

Utwórz w klasie `Fraction` konstruktor kopiujący i napisz program pokazujący działanie tego konstruktora.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.4.

Zauważmy, że obiekty `new Fraction(-3, 4)` i `new Fraction(3, -4)` reprezentują ten sam ułamek $-\frac{3}{4}$. Po zamianie obiektów na łańcuchy znaków (metodą `toString()`) otrzymamy odpowiednio `"-3/4"` i `"3/-4"`. Podobnie wyglądałaby sytuacja dla obiektów `new Fraction(2, 5)` i `new Fraction(-2, -5)` reprezentujących ułamek `2/5`. Zapis ułamka w postaci `"3/-4"` lub `"-2/-5"` nie wygląda korzystnie (lepszy będzie zapis `"-3/4"` lub `"2/5"`). Można przyjąć, że będziemy zapamiętywali zawsze dodatni mianownik, a znak licznika zadecyduje o znaku ułamka. Dodaj do klasy `Fraction` prywatną metodę, która wywołana wewnątrz konstruktora skoryguje licznik i mianownik ułamka zgodnie z przyjętą umową. Napisz program pokazujący skutki działania tej metody.


Uwaga

Metodę nazwij `correction()` (poprawka). Będzie ona przydatna podczas wyznaczania odwrotności ujemnego ułamka lub dzielenia przez taki ułamek.

Zadanie 20.5.

Zauważmy, że obiekty `new Fraction(3, 4)` i `new Fraction(15, 20)` reprezentują ten sam ułamek *3/4*. Dodaj do klasy `Fraction` publiczne metody służące do skracania ułamka (przez największy wspólny dzielnik licznika i mianownika lub inną podaną wartość) oraz publiczną metodę pozwalającą na rozszerzanie ułamka. Napisz program pokazujący działanie tych metod.


Uwaga

Metodę skracającą ułamek nazwij `reduce()` (ang. *reducing fraction* — skracać ułamek). Do realizacji tej metody niezbędna będzie prywatna metoda obliczająca najmniejszy wspólny dzielnik (ang. *Greatest Common Factor — GCF*). Proponujemy w tym przypadku użycie polskiego skrótu `nwd`. Metodę rozszerzającą ułamek nazwij `equivalent()` (ang. *equivalent fraction* — ułamek).

Zadanie 20.6.

W klasie `Fraction` utwórz metody zwracające nowy obiekt `Fraction`, będący iloczynem ułamka reprezentowanego przez ten obiekt i inny obiekt lub liczbę całkowitą. Napisz program pokazujący działanie tych metod.


Wskazówka

Dzięki możliwości przeciążania nazw metod obie metody mogą mieć tę samą nazwę `mult()` (ang. *multiplication* — mnożenie).

Zadanie 20.7.

W klasie `Fraction` utwórz metody statyczne (o nazwie `product()`, ang. *product* — iloczyn) zwracające obiekt `Fraction`, będący iloczynem dwóch ułamków, ułamka i liczby całkowitej lub dwóch liczb całkowitych. Napisz program demonstrujący działanie tych metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.8.

W klasie `Fraction` utwórz metodę zwracającą nowy obiekt `Fraction`, reprezentujący ułamek odwrotny do ułamka zawartego w obiekcie wywołującym tę metodę. Utwórz metodę statyczną o podobnej funkcjonalności. Napisz program pokazujący działanie tych metod. Wykorzystaj odwrotność do obliczenia ilorazu dwóch ułamków.


Wskazówka

Metodom nadaj nazwę `multInv()`, pochodzącą od określenia ang. *multiplicative inverse* — odwrotność (liczby).

Zadanie 20.9.

W klasie `Fraction` utwórz metody zwracające nowy obiekt `Fraction`, będący ilorazem ułamka reprezentowanego przez ten obiekt i inny obiekt lub liczbę całkowitą.


Wskazówka

Metodom nadaj nazwę `div()` (ang. *division* — dzielenie).

Zadanie 20.10.

W klasie `Fraction` utwórz metody statyczne (o nazwie `quot()`, ang. *quotient* — iloraz) zwracające obiekt `Fraction`, będący ilorazem dwóch ułamków, ułamka i liczby całkowitej lub dwóch liczb całkowitych. Napisz program demonstrujący działanie tych metod.

Zadanie 20.11.

W klasie `Fraction` utwórz metody (o nazwie `add()`, ang. *addition* — dodawanie) zwracające nowy obiekt `Fraction`, będący sumą ułamka reprezentowanego przez ten obiekt i inny obiekt lub liczbę całkowitą. Napisz program pokazujący działanie tych metod.


Wskazówka

Podczas dodawania ułamków o różnych mianownikach niezbędne jest sprowadzenie ułamków do wspólnego mianownika.

Wspólnym mianownikiem może być iloczyn mianowników, czyli dodawanie możemy wykonać według wzoru $\frac{a}{b}+\frac{c}{d}=\frac{ad}{bd}+\frac{bc}{bd}=\frac{ad+bc}{bd}$. Korzystniej będzie jednak obliczyć najmniejszą wspólną wielokrotność (*NWW(b, d)*) mianowników i tę liczbę przyjąć jako wspólny mianownik (zob. rozwiązania zadań 15.7 lub 15.9). Prywatną metodę `nww()` dodaj do klasy `Fraction`.

Zadanie 20.12.

W klasie `Fraction` utwórz metody statyczne (o nazwie `sum()`, ang. *sum* — suma) zwracające obiekt `Fraction`, będący sumą dwóch ułamków lub ułamka i liczby całkowitej. Napisz program demonstrujący działanie tych metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.13.

W klasie `Fraction` utwórz metodę zwracającą nowy obiekt `Fraction`, reprezentujący ułamek przeciwny do ułamka zawartego w obiekcie wywołującym tę metodę. Utwórz metodę statyczną o podobnej funkcjonalności. Napisz program pokazujący działanie tych metod. Wykorzystaj ułamek przeciwny do obliczenia różnicy dwóch ułamków.


Wskazówka

Metodom nadaj nazwę `addInv()`, pochodzącą od określenia ang. *additive inverse* — liczba przeciwna.

Zadanie 20.14.

W klasie `Fraction` utwórz metody (o nazwie `sub()`, ang. *subtraction* — odejmowanie) zwracające nowy obiekt `Fraction`, będący różnicą ułamka reprezentowanego przez ten obiekt i inny obiekt lub liczbę całkowitą. Napisz program pokazujący działanie tych metod.


Wskazówka

Odejmowanie możemy wykonać według wzoru $\frac{a}{b}-\frac{c}{d}=\frac{ad}{bd}-\frac{bc}{bd}=\frac{ad-bc}{bd}$ lub stosując jako wspólny mianownik *NWW(b, d)*. Proponujemy uprościć sprawę i zastąpić odejmowanie dodawaniem liczby przeciwnej: $\frac{a}{b}-\frac{c}{d}=\frac{a}{b}+\frac{-c}{d}$.

Zadanie 20.15.

W klasie `Fraction` utwórz metody statyczne (o nazwie `diff()`, ang. *difference* — różnica) zwracające obiekt `Fraction`, będący różnicą dwóch ułamków lub ułamka i liczby całkowitej. Napisz program demonstrujący działanie tych metod.

Zadanie 20.16.

Dodaj do klasy `Fraction` metody (`getNum()` i `getDen()`) pozwalające na odczytanie wartości prywatnych pól obiektu (licznika i mianownika ułamka reprezentowanego przez obiekt). Napisz program demonstrujący działanie tych metod.

Zadanie 20.17.

Dopuszcza się zmiany wartości prywatnych pól obiektu (przy użyciu metod) w celu zmiany jego wartości bez tworzenia nowej instancji obiektu. Zdefiniuj w klasie `Fraction` metody `setNum(int)`, `setDen(int)` i `setFrac(int, int)`, które będą zmieniać wartość licznika, mianownika lub jednocześnie licznika i mianownika ułamka (obiektu). Napisz program demonstrujący działanie tych metod.

Zadanie 20.18.

Metoda `Object.equals()` jest dziedziczona przez wszystkie klasy i pozwala ustalić, czy dwa obiekty są identyczne. W klasie `Fraction` za równe uznamy te obiekty, które reprezentują ten sam ułamek. Utwórz i dołącz do klasy metodę `equals()`, przesłania-

Ebookpoint.pl kopia dla: pawher@o2.pl

jącą metodę `Object.equals()`, oraz napisz program testujący poprawność działania zbudowanej metody.


Wskazówka

Do porównania ułamków wykorzystaj zależność $\frac{a}{b}=\frac{c}{d} \Leftrightarrow ad=bc$. Metoda ta zawiedzie, gdy wartość iloczynów (*ad* lub *bc*) przekroczy zakres liczb całkowitych. Bezpieczniejszym rozwiązaniem byłoby doprowadzenie obu ułamków do postaci nieskracalnej — takie ułamki będą równe wtedy i tylko wtedy, gdy będą miały równe liczniki i równe mianowniki, np. $\frac{12}{27}=\frac{4}{9}$ i $\frac{20}{45}=\frac{4}{9}$, więc $\frac{12}{27}=\frac{20}{45}$.


Uwaga

Metoda `equals()` powinna być *zwrotna*, *symetryczna* i *przechodnia* (cechy *relacji równoważności*). Ponadto powinna być *spójna*, czyli wielokrotne wywoływanie `x.equals(y)` musi konsekwentnie zwracać tę samą wartość (albo `true`, albo `false`), o ile nie została zmodyfikowana żadna dana wykorzystywana do porównywania (w naszym przypadku *skracanie* lub *rozszerzanie ułamka* nie może mieć wpływu na wynik porównania). Jeśli `x` jest niepustą referencją (wskazuje obiekt), to `x.equals(null)` musi zwracać wartość `false`.

Jeśli w klasie przedefiniujemy metodę `equals()`, to należy również przedefiniować metodę `hashCode()`. Jest to konieczne do prawidłowej współpracy klasy z kolekcjami korzystającymi z kodowania mieszającego (np. `HashSet`).

Zadanie 20.19.

Dodaj do klasy `Fraction` metody zwracające wartość dziesiętną ułamka reprezentowanego przez obiekt. Napisz program demonstrujący działanie tych metod.


Wskazówka

Podziel licznik przez mianownik, pamiętając o tym, aby co najmniej jedną z liczb rzutować przed wykonaniem dzielenia na typ zmiennoprzecinkowy. Sugerowane nazwy metod to `doubleValue()` i `floatValue()`; dla metod statycznych: `toDouble(Fraction)`, `toFloat(Fraction)`.

Zadanie 20.20.

Ułamek może być przedstawiony jako łańcuch znaków w postaci `"4/7"` (ułamek zwykły), `"5"` (liczba całkowita odpowiadająca ułamkowi $^{5}/_{1}$), `"2.45"` (ułamek dziesiętny odpowiadający ułamkowi $^{245}/_{100}$) lub `"2,45"` (według zasad zapisu obowiązujących w Polsce). Zbuduj konstruktor, który na podstawie łańcucha znaków utworzy odpowiedni obiekt klasy `Fraction` lub zgłosi wyjątek, gdy łańcuch znaków nie będzie przedstawiać ułamka. Napisz program demonstrujący działanie tego konstruktora.

Zadanie 20.21.

Zbuduj kilka metod statycznych o nazwie `valueOf()` z jednym parametrem (typu `float`, `double`, `int`, `String`) lub dwoma parametrami (typu `int`), zwracających obiekt `Fraction`

Ebookpoint.pl kopia dla: pawher@o2.pl

reprezentujący ułamek o wartości odpowiadającej podanemu parametrowi. Napisz program demonstrujący działanie tych metod.

W kolejnych zadaniach (20.22 – 20.27) wykorzystamy obiekty i metody klasy `Fraction`, uzyskując wyniki obliczeń w postaci dokładnej, wyrażonej ułamkami zwykłymi.

Zadanie 20.22.

Określamy nieskończony ciąg liczbowy w następujący sposób: $a_1 = 1$, $a_2 = 1 + \frac{1}{1}$, $a_3 = 1 + \frac{1}{1 + \frac{1}{1}}$, $a_4 = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}}$ .... Korzystając z obiektów i metod klasy `Fraction`, napisz program obliczający 15 początkowych wyrazów ciągu. Ile maksymalnie wyrazów tego ciągu mógłbyś w tym programie obliczyć?

Zadanie 20.23.

Korzystając z obiektów i metod klasy `Fraction`, napisz program obliczający sumy częściowe nieskończonego szeregu $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$.

Zadanie 20.24.

Korzystając z obiektów i metod klasy `Fraction`, napisz program obliczający sumy częściowe nieskończonego szeregu $1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \frac{1}{16} - \ldots$.

Zadanie 20.25.

Napisz program rozwiązujący równanie o postaci $ax + b = 0$ w zbiorze liczb wymiernych, posługując się obiektami i metodami klasy `Fraction`.

Zadanie 20.26.

Napisz program rozwiązujący metodą wyznaczników układ dwóch równań liniowych z dwiema niewiadomymi w zbiorze liczb wymiernych. Wykorzystaj obiekty i metody klasy `Fraction`. Utwórz metodę `det()`, która będzie obliczać wyznacznik, oraz metodę `inputFraction()` do wprowadzania z klawiatury wartości obiektów klasy `Fraction`.

Wygodnie jest zbudować metodę `inputFraction()` z jednym parametrem typu `String`, będącym opisem poprzedzającym wprowadzanie (tzw. znak zachęty). Wprowadzenie wartości obiektu `a` może wyglądać tak: `Fraction a = inputFraction ("a = ");` — w konsoli pojawi się napis `a =` i migający kursor; użytkownik wpisuje wartość i naciska klawisz *Enter*. Z konsoli pobierzemy tekst i zamienimy go na obiekt `Fraction`, stosując metodę `valueOf()`. Pozwoli to na poprawne zinterpretowanie danych o postaci `"2,45"`, `"2.45"`, `"-5"` lub `"2/3"`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.27.

Pierwiastek drugiego stopnia z liczby nieujemnej `a` możemy obliczyć ze wzoru iteracyjnego $x = \frac{1}{2}\left(x + \frac{a}{x}\right)$, biorąc wartość początkową $x = 1$. Korzystając z obiektów i metod klasy `Fraction`, napisz program obliczający przybliżenie liczby $\sqrt{5}$ w postaci ułamka zwykłego.


Wskazówka

Ten proces iteracyjny jest szybko zbieżny i 4 – 5 iteracji wystarczy do wyznaczenia wyniku. Większa liczba iteracji nie jest możliwa do wykonania przy użyciu klasy `Fraction` (zbyt szybko rosnący licznik i mianownik ułamka).

21. Klasa opakowująca Angle — miara kąta i funkcje trygonometryczne

Zadanie 21.1.

Utwórz klasę `Angle` zawierającą jedno pole prywatne `x` typu `double` przechowujące miarę kąta podaną w radianach. W klasie zbuduj konstruktor z jednym parametrem `double` (miara kąta w radianach) i sześć metod zwracających wartości funkcji trygonometrycznych kąta reprezentowanego przez obiekt. Napisz program demonstrujący działanie konstruktora i utworzonych metod.


Uwaga

Oprócz powszechnie znanych funkcji trygonometrycznych *sinus*, *cosinus* *tangens* i *cotangens* w klasie `Angle` uwzględnimy mniej popularne funkcje *secans* ($\sec x = \frac{1}{\cos x}$) i *cosecans* ($\csc x = \frac{1}{\sin x}$).

Zadanie 21.2.

Dołącz do klasy `Angle` publiczne metody (`radian()` i `degree()`) zwracające liczbę typu `double`, miarę kąta reprezentowanego przez obiekt wyrażoną w *radianach* i *stopniach*. Napisz program demonstrujący działanie tych metod.

Zadanie 21.3.

Dołącz do klasy `Angle` publiczną metodę `toString()`, która będzie zwracać łańcuch znaków (o postaci `45°30'15"`) przedstawiający miarę kąta reprezentowanego przez obiekt wyrażoną w stopniach, minutach i sekundach. Napisz program demonstrujący działanie tej metody.

Ebookpoint.pl kopia dla: pawher@o2.pl

> Metoda `toString()` przesłoni metodę o tej samej nazwie, dziedziczoną z klasy `Object`. Metodę tę należy zdefiniować z adnotacją `Override` (`@Override`).

Zadanie 21.4.

Dołącz do klasy `Angle` trzy konstruktory — z jednym, dwoma lub trzema parametrami typu `int` — umożliwiające zbudowanie obiektu reprezentującego kąt o podanej liczbie stopni, stopni i minut lub stopni, minut i sekund. Ogranicz liczbę stopni do zakresu od 0° do 360°, a liczbę minut i sekund do zakresu od 0 do 60. Napisz program demonstrujący działanie tych konstruktorów.

Zadanie 21.5.

Ciąg znaków o postaci `105°30'15"` przedstawia miarę kąta w stopniach, minutach i sekundach kątowych. Utwórz konstruktor w klasie `Angle`, który zbuduje obiekt reprezentujący kąt o podanej w ten sposób mierze. Napisz program demonstrujący działanie tego konstruktora.

Zadanie 21.6.

Utwórz w klasie `Angle` sześć metod o nazwie `setOfXxx`, gdzie `Xxx` oznacza nazwę funkcji trygonometrycznej, ustawiających miarę kąta reprezentowanego przez obiekt na podstawie podanej wartości odpowiedniej funkcji (metody te odpowiadają funkcjom odwrotnym do funkcji trygonometrycznych). Metody zmieniają wartość obiektu, ale nie zwracają żadnej wartości. Napisz program demonstrujący działanie tych metod.

Zadanie 21.7.

Na płaszczyźnie z układem współrzędnych często potrzebujemy wyznaczyć miarę kąta nachylenia prostej przechodzącej przez początek układu współrzędnych i punkt *P(x, y)* do osi *OX*. Utwórz w klasie `Angle` metodę o nazwie `setOfPoint()`, ustawiającą miarę tego kąta jako wartość obiektu. Napisz program demonstrujący działanie metody `setOfPoint()`.

Zadanie 21.8.

Utwórz w klasie `Angle` konstruktor z dwoma parametrami typu `double`, tworzący nowy obiekt reprezentujący miarę kąta nachylenia prostej przechodzącej przez początek układu współrzędnych i punkt *P(x, y)* do osi *OX*. Napisz program demonstrujący działanie tego konstruktora.

Zadanie 21.9.

Zbuduj w klasie `Angle` statyczne funkcje o nazwie `valueOf()`, o różnych parametrach (takich samych jak konstruktory zbudowane w zadaniach 21.1, 21.4, 21.5 i 21.8), zwracające obiekt klasy `Angle` odpowiadający kątowi o podanych parametrach. Napisz program demonstrujący działanie tych metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 21.10.

Na miarach kątów możemy wykonywać dodawanie i odejmowanie. Dodaj do klasy `Angle` metody `add()` (`sub()`), które będą zwracać nowy obiekt, będący sumą (różnicą) miary kąta reprezentowanego przez obiekt wywołujący metodę i miary kąta w obiekcie podanym jako parametr. Napisz program demonstrujący działanie tych metod.

Zadanie 21.11.

Dołącz do klasy `Angle` statyczne metody `sum()` (`diff()`), które będą obliczać sumę (różnicę) dwóch obiektów podanych jako parametry (miar kątów reprezentowanych przez te obiekty). Wynik powinien być obiektem klasy `Angle`. Napisz program demonstrujący działanie tych metod.

Zadanie 21.12.

Miary kątów można mnożyć (dzielić) przez liczbę. Napisz metodę `mult()` (`div()`), która będzie zwracać nowy obiekt, reprezentujący kąt o mierze będącej iloczynem (ilorazem) miary kąta reprezentowanego przez obiekt i liczby podanej jako parametr (całkowitej lub zmiennoprzecinkowej). Napisz program demonstrujący działanie tych metod.

Zadanie 21.13.

Napisz metodę `prod()` (`quot()`), która będzie zwracać obiekt klasy `Angle`, reprezentujący kąt o mierze będącej iloczynem (ilorazem) miary kąta (obiektu) podanego jako parametr i liczby podanej jako drugi parametr. Napisz program demonstrujący działanie tych metod.

Zadanie 21.14.

Dołącz do klasy `Angle` statyczne pola przechowujące stałe wartości obiektów reprezentujących wybrane kąty: `RIGHT_ANGLE` (kąt prosty), `STRAIGHT_ANGLE` (kąt półpełny) i `FULL_ANGLE` (kąt pełny), `RADIAN` (1 radian), `DEGREE` (1 stopień), `ARCMINUTE` (1 minuta kątowa) i `ARCSECOND` (1 sekunda kątowa). Napisz program pokazujący wartości tych stałych.

> Stałe w Javie są właściwie zmiennymi deklarowanymi ze słowem kluczowym `final`. Wartości stałych w czasie działania programu nie możemy zmienić.

Zadanie 21.15.

Dodaj do klasy `Angle` metodę `compl()` (ang. *complementary angle* — kąt dopełniający), zwracającą obiekt reprezentujący dopełnienie kąta (obiektu) podanego jako parametr, i metodę `suppl()` (ang. *supplementary angle* — kąt przyległy), zwracającą obiekt reprezentujący miarę kąta przyległego do kąta (obiektu) podanego jako parametr. Napisz program demonstrujący działanie tych metod.

Klasa `Angle` jest już dostatecznie rozbudowana. W kilku kolejnych zadaniach pokażemy przykłady wykorzystania pochodzących z niej obiektów i metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 21.16.

Utwórz metodę `inputAngle()` z jednym parametrem typu `String` (przeznaczonym do opisania wprowadzanej wielkości), zwracającą obiekt `Angle` reprezentujący kąt o mierze podanej w stopniach. Metoda ta powinna poprawnie interpretować takie dane: liczba całkowita stopni, liczba rzeczywista (ułamek dziesiętny) bez podawania symbolu stopnia oraz ciąg znaków zawierających stopnie, minuty i sekundy kątowe z odpowiednimi symbolami jednostek (np. `24°30'15"`).

Napisz program rozwiązujący następujące zadanie: W trójkącie równoramiennym kąt przy wierzchołku ma miarę α. Oblicz miarę kąta β przyległego do podstawy tego trójkąta. Sprawdź obliczenia, dodając miary wszystkich kątów wewnętrznych trójkąta.


Wskazówka

Podczas wprowadzania danych symbol stopnia (°) możesz uzyskać, wpisując *Alt+0176* (przytrzymaj lewy klawisz *Alt* i wprowadź kod *0176* z klawiatury numerycznej). Symbole minuty i sekundy to apostrof i cudzysłów (dostępne bezpośrednio na klawiaturze). Miary kątów spełniają zależność $\alpha + 2\beta = 180°$.

Zadanie 21.17.

Napisz program rozwiązujący zadanie: W trójkącie równoramiennym kąt przyległy do podstawy ma miarę β. Oblicz miarę kąta α leżącego naprzeciw podstawy tego trójkąta. Sprawdź obliczenia, dodając miary wszystkich kątów wewnętrznych trójkąta.

Zadanie 21.18.

Napisz program rozwiązujący zadanie: Oblicz miary kątów w trójkącie równoramiennym o podstawie *a* i ramieniu długości *b*. Sprawdź obliczenia, dodając miary wszystkich kątów wewnętrznych trójkąta.

Zadanie 21.19.

Napisz program rozwiązujący zadanie: Oblicz miary kątów w trójkącie równoramiennym o podstawie *a* i wysokości (opuszczonej na podstawę) *h*. Sprawdź obliczenia, dodając miary wszystkich kątów wewnętrznych trójkąta.

Zadanie 21.20.

Napisz program rozwiązujący zadanie: Oblicz miary kątów w trójkącie o bokach *a*, *b* i *c*. Sprawdź obliczenia, dodając miary wszystkich kątów wewnętrznych trójkąta.

22. Liczby rzymskie i klasa Roman

Zadanie 22.1.

Napisz program zamieniający liczby naturalne podawane z klawiatury w postaci dziesiętnej na liczby zapisane w systemie rzymskim. Warunkiem zakończenia programu powinno być podanie liczby 0.

Ebookpoint.pl kopia dla: pawher@o2.pl

Stosując symbole *I*, *V*, *X*, *L*, *C*, *D* i *M* oraz reguły zapisu liczb rzymskich, możemy zapisać liczbę naturalną z zakresu od *0* do *3999*.

Zadanie 22.2.

Napisz program odczytujący wartość dziesiętną liczby zapisanej znakami rzymskimi. Dane wprowadzamy z klawiatury. Warunkiem zakończenia programu powinno być wprowadzenie pustego łańcucha znaków.

Zadanie 22.3.

Utwórz metodę statyczną `decToRoman()`, która będzie zamieniać liczbę naturalną z zakresu od *1* do *3999* na liczbę w zapisie rzymskim. Napisz program demonstrujący działanie tej metody.

Skorzystaj z tablic zbudowanych w rozwiązaniu zadania 22.1.

Zadanie 22.4.

W procesie kodowania liczb rzymskich stosujemy dwie tablice — tablicę z wartościami wybranych liczb rzymskich i tablicę z odpowiadającymi im wartościami dziesiętnymi. Utwórz klasę `RN` (ang. *roman numbers*), która będzie zawierać dwa pola: pole typu `String` (symbol rzymski) oraz pole typu `int` (wartość dziesiętna tej liczby rzymskiej). Klasa powinna udostępniać jedynie tablicę z parami wartości `("M", 1000)`, `("CM", 900)` itd. Zmodyfikuj metodę `decToRoman()` (z zadania 22.3) tak, aby wykorzystywała tablicę z klasy `RN`.

Zadanie 22.5.

Utwórz metodę statyczną `romanToDec()`, która będzie zamieniać liczbę zapisaną w systemie rzymskim na liczbę dziesiętną. Napisz program demonstrujący działanie tej metody.

Skorzystaj z rozwiązań zadań 22.2 i 22.4.

Zadanie 22.6.

Utwórz klasę `Roman`, która będzie mieć jedno pole typu `int` zawierające liczbę naturalną z zakresu od *1* do *3999*. W klasie tej zbuduj konstruktory tworzące obiekt na podstawie parametru będącego liczbą całkowitą lub łańcuchem znaków przedstawiającym liczbę w zapisie rzymskim. Dołącz do klasy inne metody przydatne do pracy z liczbami rzymskimi. Napisz program demonstrujący możliwości tej klasy.

Skorzystaj z rozwiązań zadań 22.1 – 22.5 oraz doświadczeń zdobytych podczas tworzenia klasy `Angle`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 22.7.

Korzystając z klasy `Roman`, napisz aplikację sprawdzającą umiejętność odczytywania wartości liczb rzymskich. Liczba pytań powinna być z góry ustalona, a wartości liczb powinny być losowane z podanego zakresu.

> Do losowania liczb można użyć metod z klasy `MyRandomArray`.

Zadanie 22.8.

Korzystając z klasy `Roman`, napisz aplikację sprawdzającą umiejętność zapisywania liczb w systemie rzymskim. Liczba pytań powinna być z góry ustalona, a wartości liczb powinny być losowane z podanego zakresu.

23. Trójmian kwadratowy i klasa QuadratPoly

Zadanie 23.1.

Zbuduj klasę `QuadratPoly` (ang. *quadratic polynomial* — trójmian kwadratowy) umożliwiającą rozwiązywanie różnych zadań dotyczących trójmianu kwadratowego ($ax^2 + bx + c, \quad a \neq 0$) o współczynnikach całkowitych. W klasie tej utwórz konstruktor z trzema parametrami i metodę `value()` z jednym parametrem, zwracającą wartość trójmianu kwadratowego dla podanego argumentu. Napisz program demonstrujący działanie konstruktora i zdefiniowanej metody.

Zadanie 23.2.

Dodaj do klasy `QuadratPoly` pole `delta` przechowujące wartość wyróżnika trójmianu kwadratowego ($\Delta = b^2 - 4ac$) i metodę `getDelta()` zwracającą wartość wyróżnika trójmianu. Obliczenie wyróżnika powinno być zrealizowane w konstruktorze podczas tworzenia nowego obiektu. Napisz program demonstrujący działanie tej metody.

Zadanie 23.3.

Utwórz i dołącz do klasy `QuadratPoly` metodę `toString()` zwracającą trójmian w postaci tekstowej, np. `"2x^2-x+3"`. Napisz program demonstrujący działanie tej metody.

Zwróć uwagę na sposób przedstawiania współczynników `-1`, `1` i `0` — pomiń liczby `-1` i `1` przed wyrażeniem `x^2` lub `x`, pozostawiając odpowiedni znak; nie wypisuj wyrażenia `0x` oraz wyrazu wolnego `0`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 23.4.

Utwórz w klasie `QuadratPoly` metodę `sgnDelta()` zwracającą znak wyróżnika trójmianu. Od znaku wyróżnika zależy liczba pierwiastków trójmianu. Napisz program informujący o liczbie rzeczywistych pierwiastków trójmianu kwadratowego.


Wskazówka

> Funkcja `sgn` (łac. *signum* — znak) zwraca `-1`, gdy argument jest ujemny, `1`, gdy argument jest dodatni, oraz `0` dla wartości zero.

Zadanie 23.5.

Dołącz do klasy `QuadratPoly` metody `getX1()` i `getX2()` zwracające pierwiastki trójmianu kwadratowego reprezentowanego przez obiekt. Jeśli trójmian nie ma pierwiastków rzeczywistych, to metody mogą zwracać wartość `NaN`. Korzystając z tych metod, napisz program rozwiązujący równanie kwadratowe o współczynnikach całkowitych, wprowadzonych z klawiatury.

Zadanie 23.6.

Wykresem trójmianu kwadratowego $ax^2 + bx + c, \quad a \neq 0$ jest parabola. Charakterystycznym punktem paraboli jest jej wierzchołek, czyli punkt o współrzędnych (p, q), gdzie $p = -\frac{b}{2a}$ i $q = -\frac{\Delta}{4a}$. Dołącz do klasy `QuadratPoly` metody `getP()` i `getQ()` zwracające współrzędne wierzchołka paraboli. Napisz program demonstrujący działanie tych metod.

Zadanie 23.7.

Dołącz do klasy `QuadratPoly` metodę `isAPositve()` zwracającą wartość `true`, gdy `a` jest dodatnie, i `false`, gdy `a` jest ujemne. Napisz program wyznaczający zbiór wartości dla podanego trójmianu kwadratowego.


Wskazówka

> Jeśli *a > 0*, to zbiorem wartości trójmianu $ax^2 + bx + c$ jest przedział $\langle q, +\infty)$, a dla *a < 0* — przedział $(-\infty, q\rangle$.

Zadanie 23.8.

Napisz program, który dla danego trójmianu kwadratowego określi jego wartość ekstremalną (minimum lub maksimum). Dane trójmianu użytkownik wprowadzi z klawiatury.

Zadanie 23.9.

Dołącz do klasy `QuadratPoly` metodę `getVertex()` (ang. *vertex* — wierzchołek), która będzie zwracać współrzędne wierzchołka paraboli będącej wykresem trójmianu reprezentowanego przez obiekt. Wynik powinien być podany w postaci łańcucha znaków. Napisz program demonstrujący działanie tej metody.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 23.10.

Dołącz do klasy `QuadratPoly` metodę `getCodomain()` (ang. *codomain* — przeciwdziedzina funkcji), która będzie zwracać zbiór wartości trójmianu (przedział liczbowy) zapisany w postaci łańcucha znaków. Napisz program demonstrujący działanie tej metody.

Zadanie 23.11.

Napisz program sporządzający opis przebiegu zmienności funkcji kwadratowej $f(x) = ax^2 + bx + c, \quad a \neq 0$.


Wskazówka

Opis zredaguj w punktach. Umieść w opisie dziedzinę funkcji, zbiór wartości, monotoniczność, ekstremum, miejsca zerowe i krótkie objaśnienie wykresu.

Zadanie 23.12.

Napisz program rozwiązujący nierówność kwadratową $ax^2 + bx + c > 0, \quad a \neq 0$.

Zadanie 23.13.

Napisz program rozwiązujący nierówność kwadratową $ax^2 + bx + c \geq 0, \quad a \neq 0$.

Zadanie 23.14.

Dołącz do klasy `QuadratPoly` metodę `mult()` z parametrem będącym liczbą całkowitą różną od zera, która zwraca nowy obiekt, iloczyn tej liczby i trójmianu reprezentowanego przez obiekt wywołujący tę metodę. Napisz program demonstrujący jej działanie.

Zadanie 23.15.

Dołącz do klasy `QuadratPoly` metodę `solutionOfQE()` (ang. *solution* — rozwiązanie), która będzie zwracać rozwiązanie równania kwadratowego w postaci zbioru pierwiastków (łańcuch znaków), np. `{-2, 0.5}`, `{-3}` lub `{}` (odpowiednio: dwa różne pierwiastki, pierwiastek dwukrotny lub brak pierwiastków rzeczywistych — zbiór pusty). Napisz program demonstrujący działanie tej metody.


Uwaga

W zestawie znaków dla konsoli nie znajdziemy symbolu zbioru pustego (∅), więc proponujemy zastąpienie go pustą parą nawiasów klamrowych {}.

Zadanie 23.16.

Dołącz do klasy `QuadratPoly` dwie prywatne metody: `solutionQIe1()` i `solutionQIe2()`, które będą zwracać zbiór rozwiązań nierówności kwadratowej $ax^2 + bx + c > 0, \quad a \neq 0$ (pierwsza metoda) i $ax^2 + bx + c \geq 0, \quad a \neq 0$ (druga metoda). Wynik powinien być podany w postaci łańcucha znaków. Korzystając z tych metod, zbuduj publiczną metodę `solutionQIe()` (ang. *quadratic inequality* — nierówność kwadratowa) z jednym

Ebookpoint.pl kopia dla: pawher@o2.pl

parametrem (typu `String`) o postaci `"<"`, `"<="`, `">"` lub `">="`, określającym typ nierówności. Napisz program demonstrujący rozwiązywanie nierówności kwadratowych.


Wskazówka

> Wykorzystaj zbudowane metody prywatne i fakt, że nierówność $ax^2 + bx + c < 0$ ($ax^2 + bx + c \le 0$) jest równoważna nierówności $-ax^2 - bx - c > 0$ ($-ax^2 - bx - c \ge 0$).

Zadanie 23.17.

Napisz program rozwiązujący równanie dwukwadratowe o współczynnikach całkowitych ($ax^4 + bx^2 + c = 0, \quad a \neq 0$).

Zadanie 23.18.

Trójmiany kwadratowe można dodawać lub odejmować. Wynik nie zawsze jednak będzie trójmianem kwadratowym. Dołącz do klasy metody `add()` i `sub()`, które będą zwracać nowy obiekt `QuadratPoly`, odpowiednio sumę i różnicę obiektu wywołującego metodę i obiektu przekazanego jako parametr.

Zadanie 23.19.

Zbuduj klasę `FloatQP` lub `DoubleQP` (ang. *quadratic polynomial* — trójmian kwadratowy), która będzie umożliwiać rozwiązywanie różnych zadań dotyczących trójmianu kwadratowego ($ax^2 + bx + c, \quad a \neq 0$) o współczynnikach zmiennoprzecinkowych (typu `float` lub `double`). Jeśli wynik nie jest obiektem klasy, to zgłoś wyjątek. Do klasy dodaj metodę `div()` zwracającą nowy obiekt, wynik dzielenia trójmianu przez liczbę różną od zera. Napisz program demonstrujący działanie konstruktora i wybranych metod.

24. Liczby zespolone — budujemy klasę Complex

Brak rozwiązania równania kwadratowego $x^2 = 1$ w zbiorze liczb rzeczywistych doprowadził do odkrycia jednostki urojonej i — takiej, że $i^2 = -1$ — oraz liczb urojonych (będących iloczynami liczby rzeczywistej i jednostki urojonej — ai). W konsekwencji doprowadziło to do powstania *teorii liczb zespolonych*.

Liczbę zespoloną możemy zapisywać w postaci pary liczb rzeczywistych (a, b), gdzie a oznacza część rzeczywistą liczby zespolonej, b — część urojoną. Tę parę liczb możemy interpretować jako współrzędne punktu na płaszczyźnie (tzw. *płaszczyzna zespolona*).

Inną możliwością jest zapis $a+bi$, czyli *postać kanoniczna liczby zespolonej*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.1.

Zbuduj klasę `Complex`, której obiekty będą reprezentowały liczby zespolone. Klasa powinna posiadać dwa prywatne pola typu `double`, co najmniej jeden konstruktor z dwoma parametrami (typu `double`), metody ustawiające wartość części rzeczywistej i urojonej oraz możliwość przedstawienia liczby w postaci algebraicznej (kanonicznej). Napisz aplikację pokazującą działanie konstruktorów metod tej klasy.


Uwaga

Prywatne pola klasy `Complex` oznaczmy identyfikatorami `re` (ang. *real* — część rzeczywista liczby zespolonej) i `im` (ang. *imaginary* — część urojona). Metody `setRe(double x)` i `setIm(double y)` powinny umożliwić zmianę wartości liczby zespolonej reprezentowanej przez obiekt, natomiast metoda `toString()` zwróci łańcuch znaków reprezentujący postać kanoniczną liczby.

Zadanie 24.2.

Napisz aplikację rozwiązującą równanie kwadratowe o współczynnikach rzeczywistych w zbiorze liczb zespolonych.


Wskazówka

Algorytm rozwiązywania równania jest taki sam jak w zbiorze liczb rzeczywistych. Jedynie w przypadku, gdy wyróżnik jest ujemny ($\Delta < 0$), rozwiązaniem równania będzie para liczb zespolonych, ponieważ $\sqrt{\Delta} = \sqrt{-\Delta}\,i$.

Zadanie 24.3.

Dołącz do klasy `Complex` metody zwracające sumę (`add()` — ang. *addition*) i różnicę (`sub()` — ang. *subtraction*) wartości zespolonej reprezentowanej przez obiekt i wartości podanej jako parametr. Napisz aplikację pokazującą działanie tych metod.

Zadanie 24.4.

Dołącz do klasy `Complex` metody zwracające liczbę zespoloną przeciwną (`opp()` — ang. *opposite* lub *additive inverse*), sprzężoną (`conj()` — ang. *conjugate*) i odwrotną (`rec()` — ang. *reciprocal* lub *multiplicative inverse*) do wartości zespolonej reprezentowanej przez obiekt. Napisz aplikację pokazującą działanie tych metod.

Zadanie 24.5.

Dołącz do klasy `Complex` metody zwracające iloczyn (`mult()` — ang. *multiplication*) i iloraz (`div()` — ang. *division*) wartości zespolonej reprezentowanej przez obiekt i wartości podanej jako parametr. Napisz aplikację pokazującą działanie tych metod.

Zadanie 24.6.

Dodaj do klasy `Complex` stałe `ZERO` (zero), `ONE` (jeden), `I` (jednostka urojona *i* — ang. *imaginary unit*). Napisz aplikację pokazującą znaczenie tych stałych.

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Stała `ZERO` jest elementem neutralnym dla dodawania, a stała `ONE` stanowi element neutralny dla mnożenia. Korzystając ze stałej `I`, możemy pokazać potęgi jednostki urojonej.

Zadanie 24.7.

Dołącz do klasy `Complex` metody zwracające bezwzględną wartość (moduł) liczby zespolonej (`abs()` — ang. *absolute value*) i argument główny liczby zespolonej (`arg()`) reprezentowanej przez obiekt. Napisz aplikację pokazującą działanie tych metod.

Zadanie 24.8.

Dołącz do klasy `Complex` metody zwracające część rzeczywistą (`getRe()`) i część urojoną (`getIm()`) liczby zespolonej reprezentowanej przez obiekt. Napisz aplikację pokazującą działanie tych metod.

Zadanie 24.9.

Dodaj do klasy `Complex` bezparametrowe metody `print()` i `println()` wyświetlające postać kanoniczną liczby zespolonej w konsoli. Zbuduj te metody również z jednym parametrem typu `String`. Podany łańcuch znaków powinien poprzedzać wyświetlaną liczbę zespoloną. Napisz aplikację pokazującą działanie tych metod.


Uwaga

Zaproponowane metody skrócą nam zapis wielu rozwiązań zadań. Różnica w działaniu metody `println()` i `print()` jest oczywista i w naszej implementacji nie powinna odbiegać od przyjętego standardu.

Zadanie 24.10.

Dołącz do klasy `Complex` konstruktor kopiujący oraz metodę umożliwiającą porównywanie obiektu wywołującego tę metodę z innym obiektem podanym jako parametr. Napisz aplikację pokazującą porównywanie obiektów.

Zadanie 24.11.

Dołącz do klasy `Complex` konstruktor tworzący obiekt na podstawie łańcucha znaków przedstawiającego liczbę zespoloną w postaci `"-3.4+2.75i"`. Napisz aplikację pokazującą działanie tego konstruktora.


Uwaga

Konstruktor powinien poprawnie przetwarzać łańcuchy o postaci: `"1"`, `"-2.5"`, `"+3.5"`, `"i"`, `"+i"`, `"-i"`, `"2+3i"`, `"-2.3+3i"`, `"2-3i"`, `"-2.3+3i"`, `"+2-3.7i"`, `"+2+3.7i"`, `"1+i"`, `"2.1-i"`. Aby zbytnio nie komplikować kodu, zrezygnujemy z zapisywania liczb w notacji naukowej.

Zadanie 24.12.

Zbuduj w klasie `Complex` metodę statyczną `parseComplex()` zamieniającą łańcuch znaków na liczbę zespoloną. Napisz aplikację pokazującą działanie tej metody.

Ebookpoint.pl kopia dla: pawher@o2.pl


Wskazówka

Zastosuj konstruktor zbudowany w rozwiązaniu zadania 24.11.

Zadanie 24.13.

Utwórz poza klasą `Complex` funkcję `power()` obliczającą potęgę o wykładniku całkowitym dla dowolnej liczby zespolonej. Napisz aplikację pokazującą działanie tej funkcji.

Zadanie 24.14.

Dodaj opracowaną w zadaniu 24.13 funkcję `power()` do klasy `Complex` jako metodę statyczną oraz niestatyczną. Napisz aplikację pokazującą działanie metod `power()`.


Wskazówka

W metodzie statycznej podstawa potęgi przekazywana jest jako parametr wywołania, natomiast w metodzie niestatycznej podstawą potęgi stanie się wartość obiektu wywołującego tę metodę (wykorzystaj słowo kluczowe `this`).

Zadanie 24.15.

Dołącz do klasy `Complex` metody obliczające kwadrat liczby zespolonej (`sqr()`) i pierwiastek kwadratowy (`sqrt()`) z liczby zespolonej. Napisz aplikację pokazującą działanie tych metod.


Uwaga

W zbiorze liczb zespolonych określamy dwa pierwiastki drugiego stopnia. Są one liczbami przeciwnymi. Podczas rozwiązywania zadań należy o tym pamiętać. Można utworzyć metodę `nextSqrt()`, wyznaczającą drugi pierwiastek z liczby zespolonej, lub podać w metodzie `sqrt()`parametr, wskazujący, który z tych pierwiastków chcemy obliczyć.

Zadanie 24.16.

Dołącz do klasy `Complex` metody `printf()` i `printlnf()`, które będą wyświetlać w konsoli liczbę zespoloną z określoną precyzją domyślną (6 miejsc po przecinku). Napisz program demonstrujący działanie tych metod.


Wskazówka

Zob. rozwiązanie zadania 24.9.

Zadanie 24.17.

Napisz program rozwiązujący w zbiorze liczb zespolonych równanie kwadratowe o współczynnikach zespolonych, np.: $(1-2i)z^2 + iz - 1 = 0$.

Zadanie 24.18.

Dołącz do klasy `Complex` cztery metody statyczne (`sum()`, `diff()`, `prod()` i `quot()`), które będą realizować cztery podstawowe działania (dodawanie, odejmowanie, mnożenie i dzielenie) w zbiorze liczb zespolonych.

Ebookpoint.pl kopia dla: pawher@o2.pl

Liczbę zespoloną $z = x + yi$ możemy przekształcić na postać $z = |z| \cdot (\sin\varphi + i\cos\varphi)$ nazywaną *postacią trygonometryczną liczby zespolonej* ($|z|$ — moduł, φ — argument liczby *z*).

Postać trygonometryczną liczby zespolonej możemy skojarzyć z *biegunowym układem współrzędnych* — liczbę zespoloną możemy przedstawiać jako parę (r, φ), gdzie *r* jest (promieniem) odległością punktu *P(x, y)* od początku układu współrzędnych, a φ jest kątem pomiędzy promieniem i osią *X* układu współrzędnych.

Zadanie 24.19.

Utwórz klasę `PolarComplex`, której obiekty będą reprezentowały liczby zespolone podane we współrzędnych biegunowych (ang. *polar coordinates*). Zbuduj podstawowe konstruktory klasy `PolarComplex`, metodę zamieniającą obiekt klasy `PolarComplex` na obiekt `Complex` i metodę `toString()` zamieniającą obiekt na łańcuch znaków, np. `"[r=1,500000; fi=1,000000]"`. Napisz program demonstrujący działanie tych konstruktorów i metod.


Uwaga

Podany ciąg znaków `"[r=1,500000; fi=1,000000]"` jest propozycją, z którą Czytelnik niekoniecznie musi się zgadzać. Nie ma w tym przypadku standardu zapisu liczb podobnego do postaci kanonicznej liczby zespolonej i łatwego do zapisania w postaci łańcucha znaków. *Postać trygonometryczna* liczby zespolonej $z = r(\sin\varphi + i\cos\varphi)$ i *postać wykładnicza* $z = re^{i\varphi}$ wykorzystują elementy pary (r, φ), ale do tego celu niezbyt się nadają. Pozostańmy zatem przy zaproponowanej w zadaniu postaci łańcucha.

Zadanie 24.20.

Dołącz do klasy `PolarComplex` konstruktor kopiujący i konstruktor z parametrem typu `Complex`. Napisz program demonstrujący działanie tych konstruktorów.

Zadanie 24.21.

Dołącz do klasy `Complex` konstruktor z parametrem typu `PolarComplex` i metodę `toPolarComplex()`, która będzie zwracać obiekt klasy `PolarComplex` odpowiadający obiektowi `Complex`. Napisz program demonstrujący działanie tego konstruktora i tej metody.

Zadanie 24.22.

Dołącz do klasy `PolarComplex` metodę `mult()`, która będzie zwracać iloczyn liczby zespolonej reprezentowanej przez obiekt i liczby przekazanej jako parametr. Napisz program demonstrujący działanie tej metody.

Zadanie 24.23.

Dołącz do klasy `PolarComplex` metodę `div()`, która będzie zwracać iloraz liczby zespolonej reprezentowanej przez obiekt i liczby przekazanej jako parametr. Napisz program demonstrujący działanie tej metody.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.24.

Dołącz do klasy `PolarComplex` metody `sqr()` i `cube()`, które będą zwracać kwadrat i sześcian liczby zespolonej reprezentowanej przez obiekt. Napisz program demonstrujący działanie tych metod.

Zadanie 24.25.

Dołącz do klasy `PolarComplex` metodę `power()` z jednym parametrem `n` typu całkowitego (`int`), która będzie zwracać potęgę o wykładniku `n` liczby zespolonej reprezentowanej przez obiekt. Napisz program demonstrujący działanie tej metody.

Zadanie 24.26.

Liczba zespolona ma dwa pierwiastki kwadratowe. Dołącz do klasy `PolarComplex` metodę `sqrt()` z jednym parametrem `k` typu `int`, która będzie zwracać obiekt `PolarComplex` wskazany przez parametr `k` — pierwiastek kwadratowy z liczby zespolonej. Napisz program demonstrujący działanie tej metody.

> Wyznacz resztę z dzielenia `k` przez *2* (`k%2`) — reszta *0* lub *1* wskaże numer pierwiastka.

Zadanie 24.27.

Liczba zespolona ma trzy pierwiastki sześcienne (trzeciego stopnia). Dołącz do klasy `PolarComplex` metodę `cbrt()` z jednym parametrem `k` typu `int`, która będzie zwracać obiekt `PolarComplex` wskazany przez parametr `k`, pierwiastek trzeciego stopnia z liczby zespolonej. Napisz program demonstrujący działanie tej metody.

> Wyznacz resztę z dzielenia `k` przez *3* (`k%3`) — reszta *0, 1* lub *2* wskaże numer pierwiastka.

Zadanie 24.28.

Dołącz do klasy `PolarComplex` bezparametrowe metody `sqrt()` i `cbrt()`, które będą zwracać pierwiastki kwadratowe i sześcienne z liczby reprezentowanej przez obiekt w postaci tablicy obiektów klasy `Complex`. Napisz program demonstrujący działanie tych metod.

Zadanie 24.29.

Napisz program rozwiązujący w zbiorze liczb zespolonych równanie o współczynnikach rzeczywistych:

a) $az^2 + c = 0,\ a \neq 0,$

b) $z^3 + a = 0.$

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.30.

Napisz program rozwiązujący w zbiorze liczb zespolonych równanie o współczynnikach zespolonych:

a) $az^2 + c = 0,\ a \neq 0 + 0i$,

b) $z^3 + a = 0$.

Zadanie 24.31.

Liczba zespolona ma *n* pierwiastków *n*-tego stopnia. Dołącz do klasy `PolarComplex` metodę `root()` z dwoma parametrami `n` i `k` typu `int`, która będzie zwracać obiekt `PolarComplex` wskazany przez parametr `k` — pierwiastek *n*-tego stopnia z liczby zespolonej. Napisz program demonstrujący działanie tej metody.

Zadanie 24.32.

Dołącz do klasy `PolarComplex` metodę `root()` z jednym parametrem `n` określającym stopień pierwiastka, która będzie zwracać tablicę obiektów klasy `Complex` z pierwiastkami *n*-tego stopnia z liczby zespolonej reprezentowanej przez obiekt (wywołujący metodę). Napisz program demonstrujący działanie tej metody.

Zadanie 24.33.

Napisz program wyznaczający dla liczby naturalnej *n* podanej przez użytkownika z klawiatury wszystkie pierwiastki równania:

a) $z^n + 1 = 0$,

b) $z^n + i = 0$,

c) $z^{2n} + iz^n + 1 = 0$.

Ebookpoint.pl kopia dla: pawher@o2.pl

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 4.

Czwarty krok — pliki, tablice i macierze

25. Operacje na plikach tekstowych

Zadanie 25.1.

Korzystając z obiektów i metod klasy `FileWriter`, napisz program zapisujący do pliku tekstowego *tekst.txt* jeden wiersz tekstu: *Programowanie obiektowe*.

Zadanie 25.2.

Plik tekstowy *tekst.txt* zawiera jeden wiersz tekstu: *Programowanie obiektowe*. Korzystając z obiektu i metod klasy `FileWriter`, napisz program dopisujący do tego pliku w pierwszym wierszu tekst: *w języku Java*, a w kolejnym wierszu tekst: *jest bardzo interesujące*.

W systemie Windows koniec wiersza w pliku tekstowym składa się z dwóch znaków *CR* (ang. *carriage return*, wartość ASCII *13*, znak `'\r'`) i *LF* (ang. *line feed* wartość ASCII *10*, znak `'\n'`). W systemach UNIX i Linux końcem wiersza jest *LF*, a w systemie Mac OS — znak *CR*.

Zadanie 25.3.

Korzystając z obiektu i metod klasy `FileWriter`, napisz program obliczający i zapisujący w pliku *silnia.txt* wartości `n!` (`n` silnia) dla `n` = 1, 2, …, 12. Każdy wynik zapisz w odrębnym wierszu, w postaci `12! = 479001600`.

Przypomnijmy znaczenie symbolu *n!*: *1! = 1, 2! = 1!·2 = 1·2, 3! = 2!·3 = 1·2·3* itd.

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

W tym zadaniu i zadaniach podobnych starajmy się, aby plik wyjściowy nie zawierał na końcu pustego wiersza (jeśli obecność tego ostatniego wiersza nie jest zamierzona). Taki wiersz może nam sprawić różne niespodzianki, gdy plik będzie odczytywany.

Zadanie 25.4.

Napisz program zapisujący w pliku *pierwiastki.txt* wartości pierwiastków kwadratowych i sześciennych dla liczb naturalnych od *2* do *15*. Każdy wiersz pliku powinien zawierać trzy liczby oddzielone znakami tabulatora — liczbę naturalną, pierwiastek kwadratowy z tej liczby i pierwiastek sześcienny. Pierwiastki podaj z precyzją do *8* miejsc po przecinku.


Wskazówka

Do formatowania wyników użyj metody `format()` z klasy `String`.

Zadanie 25.5.

Napisz program zapisujący w pliku tekstowym *sto.txt* sto liczb całkowitych wylosowanych z zakresu od *1* do *20*. Liczby w pliku powinny być oddzielone odstępami.

Zadanie 25.6.

Napisz program zapisujący w pliku tekstowym *dane.txt* 50 par liczb. Każda para liczb powinna być umieszczona w odrębnym wierszu. Pierwsza liczba w parze powinna być rzeczywista, dodatnia i nie większa od *10* oraz podana z dokładnością do dwóch miejsc po przecinku, druga liczba powinna być całkowita i ma należeć do przedziału $\langle 2, 8 \rangle$. Liczby w wierszu oddzielamy odstępem.


Uwaga

Po rozwiązaniu zadań 25.1 – 25.6 będziemy mieli w bieżącym folderze pięć plików: *tekst.txt*, *silnia.txt*, *pierwiastki.txt*, *sto.txt* i *dane.txt*. Zawartość tych plików będziemy odczytywali w kolejnych zadaniach.

Zadanie 25.7.

Korzystając z obiektu i metod klasy `FileReader`, napisz program odczytujący zawartość pliku tekstowego i wyświetlający jego zawartość w konsoli.


Wskazówka

Bezparametrowa metoda `read()` z klasy `FileReader` odczytuje z pliku jeden znak i zwraca jego kod, liczbę typu `int` z zakresu od *0* do *65 535* lub liczbę *–1*, gdy nie można odczytać znaku. Metoda `ready()` zwraca wartość logiczną `true`, gdy z pliku można odczytać kolejny znak, i `false` w przeciwnym wypadku.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 25.8.

Korzystając z obiektu i metod klasy `FileReader`, utwórz metodę (`readLine()`) odczytującą wiersz pliku tekstowego. Napisz program odczytujący i wyświetlający w konsoli wszystkie wiersze pliku tekstowego.


Wskazówka

Klasa `FileReader` nie zawiera metody `readLine()`, więc musisz ją sam zbudować. Plik tekstowy otwórz przed wywołaniem metody, obiekt (klasy `FileReader`) skojarzony z plikiem przekaż jako parametr do metody `readLine()`, która przeczytany z pliku wiersz tekstu zwróci w postaci łańcucha znaków.

Zadanie 25.9.

Napisz program wyświetlający w konsoli kod źródłowy programu w języku Java wraz z numerami linii. Nazwę pliku (bez rozszerzenia) użytkownik powinien podawać z klawiatury. Jeśli w określonej lokalizacji nie ma wskazanego pliku, to program powinien wyświetlić odpowiedni komunikat.


Wskazówka

Do wprowadzenia nazwy pliku z konsoli oraz do odczytania wierszy tekstu z pliku wykorzystaj obiekty klasy `BufferedReader` i metodę `readLine()` z tej klasy. Obecność pliku możesz sprawdzić, stosując metodę `exists()` z klasy `File`.

Zadanie 25.10.

Napisz program zapisujący do pliku kod źródłowy programu w języku Java wraz z numerami linii. Nazwę pliku (bez rozszerzenia) użytkownik powinien podawać z klawiatury. Jeśli w określonej lokalizacji nie ma wskazanego pliku, to program powinien wyświetlić odpowiedni komunikat. Nazwa pliku wyjściowego powinna być taka jak nazwa pliku źródłowego, rozszerzenie *java* zamienimy na rozszerzenie *txt*.


Wskazówka

Do wprowadzenia nazwy pliku z konsoli oraz do odczytania wierszy tekstu z pliku wykorzystaj obiekty klasy `Scanner`. Do zapisania pliku wyjściowego użyj metod klasy `PrintWriter`.

Zadanie 25.11.

W pliku tekstowym wpisany jest ciąg liczb całkowitych oddzielonych odstępami. Napisz program, który odczyta i wyświetli w konsoli liczby z pliku oraz obliczy ich sumę.


Wskazówka

Do testów możesz użyć pliku *sto.txt* (rozwiązanie zadania 25.5). Nie wykorzystuj jednak faktu, że znasz ilość liczb zapisanych w tym pliku.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 25.12.

W pliku tekstowym wpisany jest ciąg liczb całkowitych oddzielonych odstępami. Napisz program, który znajdzie najmniejszą liczbę w tym pliku oraz obliczy, ile razy ta liczba w tym pliku występuje.

Do testów możesz użyć pliku *sto.txt* (rozwiązanie zadania 25.5). Nie wykorzystuj jednak faktu, że znasz ilość liczb zapisanych w pliku.

Zadanie 25.13.

W pliku tekstowym zapisane są w kolejnych wierszach pary liczb — liczba zmiennoprzecinkowa i liczba całkowita. Napisz program, który odczytuje pary liczb z pliku, oblicza iloczyn każdej pary i sumuje iloczyny. Wynik obliczeń należy wypisać w konsoli i zapisać w ostatnim wierszu pliku z danymi.

Do testów możesz użyć pliku *dane.txt* (rozwiązanie zadania 25.6).

26. Tablice jednowymiarowe i wielomiany

Jednym z zastosowań tablic jednowymiarowych może być przechowywanie współczynników wielomianu.

Wielomian *n*-tego stopnia jednej zmiennej $w(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ ma $n+1$ współczynników, które możemy zapisać w tablicy `double[] a = {a0, a1, ..., an};`.

Zadanie 26.1.

Napisz program obliczający wartości wielomianu. Stopień wielomianu, współczynniki i kolejne wartości argumentu użytkownik będzie wprowadzał z klawiatury. Podanie argumentu `x = 0` będzie sygnałem do zakończenia pracy programu.

Do obliczenia wartości wielomianu wykorzystaj *schemat Hornera*.

Zadanie 26.2.

Napisz program, który obliczy i zapisze w pliku tekstowym (*wielomian.txt*) tablicę wartości wielomianu $w(x) = 2x^3 + 5x^2 - x + 3$ w przedziale $\langle -2, 3 \rangle$ z krokiem $h = 0{,}125$. Plik powinien zawierać trzy wiersze z informacjami o rozwiązywanym zadaniu, według schematu: stopień wielomianu (pierwszy wiersz), współczynniki wielomianu oddzielone odstępami w drugim wierszu (zaczynając od wyrazu wolnego) i krańce

Ebookpoint.pl kopia dla: pawher@o2.pl

przedziału oraz krok w trzecim wierszu. W kolejnych wierszach umieścimy pary liczb (argument i wartość) oddzielone odstępem.

Zadanie 26.3.

Napisz program, który obliczy i wyświetli w konsoli sumę dwóch wielomianów.

> Skorzystaj z metod dostępnych w klasie `Arrays`. Przyjmij, że wyświetlając wynik lub prezentując dane, wielomian $w(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ zapiszemy w postaci łańcucha znaków `"w = [a0, a1, ... an]"`, gdzie `a0`, `a1`, ..., `an` są liczbami wyrażającymi współczynniki wielomianu.

Zadanie 26.4.

Napisz program, który obliczy i wyświetli w konsoli różnicę dwóch wielomianów.

> Zob. wskazówkę do zadania 26.3.

Zadanie 26.5.

Napisz program, który obliczy i wyświetli w konsoli iloczyn wielomianu przez liczbę.

Zadanie 26.6.

Napisz program, który obliczy i wyświetli w konsoli iloczyn dwóch wielomianów.

Zadanie 26.7.

Napisz program, który obliczy i wyświetli w konsoli pochodną wielomianu.

Zadanie 26.8.

Napisz program, który obliczy i wyświetli w konsoli całkę nieoznaczoną (funkcję pierwotną) wielomianu.

Zadanie 26.9.

Utwórz klasę `Polynomial` (plik *Polynomial.java*) umożliwiającą wykonywanie podstawowych działań na wielomianach zapisanych w postaci tablicy współczynników. Napisz aplikację prezentującą możliwości utworzonych metod i konstruktorów.

Zadanie 26.10.

Napisz program wykonujący dzielenie wielomianu

$$w(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

przez dwumian o postaci *(x–c)*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 26.11.

Dołącz do klasy `Polynomial` dwie metody — `division()` i `remainder()` — obliczające iloraz i resztę z dzielenia wielomianu reprezentowanego przez obiekt wywołujący metodę przez dwumian *(x-c)*, gdzie liczba `c` typu `double` jest parametrem wywołania metody. Napisz program demonstrujący działanie tych metod.

Zadanie 26.12.

Napisz program obliczający całkę oznaczoną $s = \int_a^b w(x)dx$ dla wielomianu

$$w(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

Wszystkie potrzebne dane użytkownik powinien wprowadzić z klawiatury — najpierw dane wielomianu (stopień i współczynniki), a później granice całkowania.

Do wyznaczenia funkcji pierwotnej użyj metody (`integral()`) opracowanej w rozwiązaniach zadań 26.8 i 26.9, a następnie skorzystaj ze wzoru $\int_a^b w(x)dx = F(b) - F(a)$.

Zadanie 26.13.

Mając dane wszystkie pierwiastki rzeczywiste wielomianu ($x_1, x_2, ..., x_n$), napisz program wyznaczający współczynniki wielomianu.

27. Obliczenia statystyczne

Zadania 27.1 – 27.19 wykonamy dla *n*-elementowej próbki zapisanej w tablicy:

```
double[] x = {1.35, 2.45, 2.05, 1.20, 2.15, 1.70, 1.45, 1.95, 2.00, 1.65, 1.65,
2.05, 1.75, 1.25, 2.25, 1.40};
```

Czytelnik może samodzielnie zmienić zestaw danych lub sposób ich pobierania przez program — wprowadzanie danych z klawiatury lub odczytywanie z pliku. Należy zwrócić uwagę na rozbieżności pomiędzy zakresem indeksów. W tablicach w języku Java indeksowanie rozpoczynamy od *0* i kończymy na indeksie o 1 mniejszym od rozmiaru tablicy, natomiast we wzorach statystyki opisowej indeksy wartości próbki będą w granicach od 1 do *n*.

W zadaniach bardzo często będziemy mieli do czynienia z obliczaniem sumy ciągu liczb $x_1, x_2, \ldots, x_n$ (oznaczanej symbolem $\sum_{i=1}^{n} x_i$) zapisanego w tablicy `x[0]`, `x[1]`, …, `x[n-1]`. Zrealizujemy to przy użyciu instrukcji pętli:

```
double suma = 0;
for(double xi: x)
    suma += xi;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

lub:

```
double suma = 0;
for(int i = 0; i < n; ++i))
    suma += x[i];
```

Zadanie 27.1.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz najmniejszą i największą wartość w ciągu oraz rozstęp badanej cechy.

Rozstępem badanej cechy jest różnica pomiędzy wartością maksymalną i minimalną $R = x_{\max} - x_{\min}$.

Zadanie 27.2.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz średnią arytmetyczną.

Średnią arytmetyczną liczb $x_1, \ldots, x_n$ nazywamy liczbę $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Zadanie 27.3.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz średnią geometryczną.

Średnią geometryczną dodatnich liczb $x_1, \ldots, x_n$ nazywamy liczbę $\overline{g} = \sqrt[n]{\prod_{i=1}^{n} x_i}$.

Jeżeli wszystkie $x_i > 0$, to $\log \overline{g} = \frac{1}{n}\sum_{i=1}^{n} \log x_i$.

Zadanie 27.4.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz średnią harmoniczną.

Średnią harmoniczną różnych od zera liczb $x_1, \ldots, x_n$ nazywamy liczbę $\overline{h} = \left(\frac{1}{n}\sum_{i=1}^{n} \frac{1}{x_i}\right)^{-1}$, gdy $\sum_{i=1}^{n} \frac{1}{x_i} \neq 0$ (odwrotność średniej arytmetycznej odwrotności tych liczb).

Zadanie 27.5.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz średnią potęgową rzędu *r*. Obliczenia wykonaj dla $r = 2$ i $r = 3$.

Ebookpoint.pl kopia dla: pawher@o2.pl

Średnią potęgową rzędu r dodatnich liczb $x_1, \ldots, x_n$ nazywamy liczbę $\overline{p}^{(r)} = \sqrt[r]{\frac{1}{n}\sum_{i=1}^{n} x_i^r}$. Dla $r = -1$ otrzymujemy średnią harmoniczną ($\overline{p}^{(-1)} = \overline{h}$), a dla $r = 1$ średnią arytmetyczną ($\overline{p}^{(1)} = \overline{x}$).

Zadanie 27.6.

Dla podanej próbki *n*-elementowej $x_1, \ldots, x_n$ wyznacz medianę.

Posortuj tablicę z danymi i wybierz element środkowy, gdy *n* jest nieparzyste, lub oblicz średnią arytmetyczną dwóch środkowych liczb, gdy *n* jest parzyste:

$$m_e = \begin{cases} x_{\frac{n+1}{2}}, & \text{gdy } n \text{ jest nieparzyste} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1} \right), & \text{gdy } n \text{ jest parzyste} \end{cases}$$

Należy pamiętać o przesunięciu indeksu wynikającego z różnicy pomiędzy indeksami we wzorach a indeksami w tablicach.

Zadanie 27.7.

Wyznacz wartość modalną (dominantę) *n*-elementowej próbki $x_1, \ldots, x_n$.

Wartością modalną (*dominantą*, *modą*) próbki $x_1, \ldots, x_n$ o powtarzających się wartościach nazywamy najczęściej powtarzającą się wartość, o ile taka istnieje. Ponadto wartość ta nie może być wartością minimalną lub maksymalną.

Zadanie 27.8.

Oblicz wariancję *n*-elementowej próbki $x_1, \ldots, x_n$. Wykorzystaj wszystkie niżej podane wzory i porównaj uzyskane wyniki.

Wariancją s^2 (*dyspersją*) próbki $x_1, \ldots, x_n$ nazywamy średnią arytmetyczną kwadratów odchyleń wartości x_i od średniej arytmetycznej $\overline{x}$ próbki: $s^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2$.

Można zastosować wzory równoważne $s^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \overline{x}^2$ lub

$s^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - a\right)^2 - \left(\overline{x} - a\right)^2$, gdzie *a* jest dowolną stałą.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 27.9.

Oblicz odchylenie standardowe n-elementowej próbki $x_1, \ldots, x_n$.


Wskazówka

Odchylenie standardowe ***s*** próbki $x_1, \ldots, x_n$ jest równe pierwiastkowi kwadratowemu z wariancji s^2 (zob. zadanie 27.8).

Zadanie 27.10.

Oblicz odchylenie przeciętne d próbki $x_1, \ldots, x_n$ od stałej a. Obliczenia wykonaj dla $a = 2$.


Wskazówka

Odchyleniem przeciętnym ***d*** *od stałej* ***a*** próbki $x_1, \ldots, x_n$ nazywamy średnią arytmetyczną wartości bezwzględnych odchyleń poszczególnych wartości x_i od stałej a:

$$d = \frac{1}{n}\sum_{i=1}^{n}|x_i - a|.$$

Zadanie 27.11.

Oblicz odchylenie przeciętne d_1 próbki $x_1, \ldots, x_n$ od wartości średniej $\bar{x}$.

Zadanie 27.12.

Oblicz odchylenie przeciętne d_2 próbki $x_1, \ldots, x_n$ od mediany m_e.

Zadanie 27.13.

Wyznacz *kwartyl dolny* Q_1 i *kwartyl górny* Q_3 próbki $x_1, \ldots, x_n$. Oblicz odchylenie ćwiartkowe Q.


Wskazówka

Wartości uporządkowanej próbki dzielimy na dwie grupy: wartości mniejsze od mediany i medianę oraz medianę i wartości większe od mediany. Kwartylem dolnym (Q_1) jest mediana pierwszej grupy, a górnym (Q_3) mediana drugiej grupy. Odchylenie ćwiartkowe Q obliczamy ze wzoru $Q = \frac{Q_3 - Q_1}{2}$.

Zadanie 27.14.

Oblicz *moment zwykły* m_l *rzędu* l próbki $x_1, \ldots, x_n$. Obliczenia wykonaj dla $l = 2, 3$ i 4.


Wskazówka

Wzór $m_l = \frac{1}{n}\sum_{i=1}^{n} x_i^l, \quad l \in N$.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 27.15.

Oblicz *moment centralny M_l rzędu l* próbki $x_1, \ldots, x_n$. Obliczenia wykonaj dla $l = 2, 3$ i 4.

Wzór $M_l = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^l, \quad l \in N$. Z własności średniej arytmetycznej wynika, że $M_1 = 0$, natomiast M_2 jest wariancją.

Zadanie 27.16.

Oblicz *moment absolutny zwykły a_l rzędu l* próbki $x_1, \ldots, x_n$. Obliczenia wykonaj dla $l = 2, 3$ i 4.

Wzór $a_l = \frac{1}{n}\sum_{i=1}^{n}|x_i|^l, \quad l \in N$.

Zadanie 27.17.

Oblicz *moment absolutny centralny b_l rzędu l* próbki $x_1, \ldots, x_n$. Obliczenia wykonaj dla $l = 2, 3$ i 4.

Wzór $b_l = \frac{1}{n}\sum_{i=1}^{n}|x_i - \bar{x}|^l, \quad l \in N$. Absolutny moment centralny rzędu pierwszego jest odchyleniem przeciętnym od średniej arytmetycznej.

Zadanie 27.18.

Oblicz *współczynnik zmienności v* próbki $x_1, \ldots, x_n$.

Wzór $v = \frac{s}{\bar{x}} \cdot 100\%$, gdzie s jest odchyleniem standardowym, a $\bar{x}$ średnią arytmetyczną próbki.

Zadanie 27.19.

Oblicz *współczynnik nierównomierności H* próbki $x_1, \ldots, x_n$.

Wzór $H = \frac{d_1}{\bar{x}} \cdot 100\%$, gdzie d_1 jest odchyleniem przeciętnym od średniej arytmetycznej $\bar{x}$.

Ebookpoint.pl kopia dla: pawher@o2.pl

W zadaniach 27.1 – 27.19 przedstawiono podstawowe wzory związane z obliczeniami statystyki opisowej dla pojedynczej próbki, zwykle nieprzekraczającej 30 elementów. Czytelnik może samodzielnie na podstawie rozwiązań tych zadań konstruować programy mające na celu rozwiązywanie problemów z zakresu statystyki.

Zadanie 27.20.

Na podstawie problemów zawartych w zadaniach 27.1 – 27.19 utwórz klasę `Stat` zawierającą metody statyczne do obliczeń statystycznych. Napisz aplikację konsolową pokazującą działanie wybranych metod z klasy `Stat`. Sporządź dokumentację tej klasy.

Zadanie 27.21.

Na podstawie problemów zawartych w zadaniach 27.1 – 27.19 utwórz klasę `Statpr` umożliwiającą utworzenie obiektu (próbki) zawierającego metody do rozwiązywania tych problemów. Napisz aplikację konsolową pokazującą działanie wybranych metod z klasy `Statpr`. Sporządź dokumentację tej klasy.

28. Tablice wielowymiarowe i macierze

Zadanie 28.1.

Utwórz dwuwymiarową tablicę liczb całkowitych o trzech wierszach. W pierwszym wierszu tablicy umieść liczby od 1 do 10, w drugim kwadraty tych liczb, a w trzecim sześciany liczb z pierwszego wiersza. Napisz program tworzący i wyświetlający tę tablicę w konsoli.

Zadanie 28.2.

W tablicy dwuwymiarowej nie wszystkie wiersze muszą mieć ten sam rozmiar. Napisz program, który utworzy tablicę liczb całkowitych o dziesięciu wierszach. Wypełnij tablicę kolejnymi liczbami naturalnymi, zaczynając od liczby *1*. W pierwszym wierszu umieść jedną liczbę, w drugim dwie liczby, w trzecim trzy itd. — w dziesiątym dziesięć liczb. Oblicz sumy liczb w kolejnych wierszach i sumę wszystkich liczb zapisanych w tablicy. Wyświetl w konsoli tablicę liczb oraz obliczone sumy.

Zadanie 28.3.

Utwórz klasę `TInt`, która będzie zawierać metody statyczne `input()` i `print()` umożliwiające wprowadzanie danych z konsoli do tablicy lub wyświetlanie danych z tablicy w konsoli. Parametrem wywołania tych metod powinna być tablica liczb całkowitych jedno- lub dwuwymiarowa. Napisz program demonstrujący działanie tych metod.

Zadanie 28.4.

Utwórz w klasie `TInt` metodę statyczną `setRandom()`, która wypełni tablicę liczb całkowitych wartościami wylosowanymi z zakresu od *0* do *n* (liczba całkowita $n > 0$).

Ebookpoint.pl kopia dla: pawher@o2.pl

Tablicę oraz zakres wartości podaj jako parametry metody. Napisz program demonstrujący działanie tej metody.

Zadanie 28.5.

Dodaj do klasy `TInt` metodę statyczną `printf()` wyświetlającą tablicę liczb całkowitych w konsoli. Metoda ta powinna mieć dwa parametry: łańcuch formatujący i identyfikator tablicy. Napisz program demonstrujący działanie tej metody.

Zadanie 28.6.

Na podstawie zadań 28.3, 28.4 i 28.5 utwórz klasę `TDouble` z metodami statycznymi `input()`, `print()`, `printf()` i `setRandom()`, ułatwiającymi pobieranie i wypisywanie danych oraz losowe ustawianie wartości w jedno- i dwuwymiarowych tablicach liczb zmiennoprzecinkowych typu `double`. Napisz program pokazujący działanie wybranych metod z tej klasy.

Macierz jest uporządkowaną prostokątną tablicą liczb, dla której zdefiniowane są działania algebraiczne dodawania (odejmowania) i mnożenia:

- Dodawanie (odejmowanie) dwóch macierzy jest możliwe tylko dla macierzy o jednakowych liczbach kolumn (n) i wierszy (m). Suma (różnica) macierzy A i B jest macierzą C taką, że $c_{ij} = a_{ij} + b_{ij}$ ($c_{ij} = a_{ij} - b_{ij}$).
- Mnożenie macierzy jest możliwe, gdy liczba kolumn pierwszej macierzy (m) jest równa liczbie wierszy drugiej macierzy. Iloczyn macierzy A i B jest macierzą C taką, że $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$. Mnożenie macierzy nie jest przemienne.
- Zawsze określone jest mnożenie macierzy przez liczbę λ, polegające na pomnożeniu każdego elementu macierzy przez tę liczbę ($c_{ij} = \lambda a_{ij}$).

Macierze możemy przedstawiać w programach jako tablice dwuwymiarowe. Należy pamiętać, że tablice są indeksowane od zera, a indeksy macierzy rozpoczynamy od jedynki.

Zadanie 28.7.

Utwórz metodę `sum()` dodającą dwie macierze zapisane w postaci tablic dwuwymiarowych. Napisz program demonstrujący działanie metody `sum()`.


Uwaga

> W tym i kolejnych zadaniach do wprowadzania danych używaj metod z klasy `TInt` lub `TDouble`.

Zadanie 28.8.

Utwórz metodę `difference()` obliczającą różnicę dwóch macierzy zapisanych w postaci tablic dwuwymiarowych. Napisz program demonstrujący działanie metody `difference()`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.9.

Utwórz metodę `product()` obliczającą iloczyn dwóch macierzy zapisanych w postaci tablic dwuwymiarowych. Napisz program demonstrujący działanie metody `product()`.

Zadanie 28.10.

Utwórz metodę `product()` obliczającą iloczyn macierzy zapisanej w postaci tablicy dwuwymiarowych przez liczbę. Napisz program demonstrujący działanie metody `product()`.

Zadanie 28.11.

Zbuduj metodę `transp()` tworzącą macierz transponowaną z macierzy podanej jako parametr metody. Napisz program demonstrujący działanie metody `transp()`.


Wskazówka

Macierz transponowana (przestawiona) powstaje z danej macierzy poprzez zamianę jej wierszy na kolumny i kolumn na wiersze.

Zadanie 28.12.

Utwórz metodę statyczną `toDouble()` konwertującą macierz o elementach całkowitych na macierz o elementach zmiennoprzecinkowych. Napisz program demonstrujący działanie tej metody. Dołącz ją do klasy `TInt`.

Zadanie 28.13.

Dołącz do klasy `TDouble` metodę statyczną o nazwie `valueOf()`, zwracającą tablicę (macierz) z elementami typu `double` o elementach odpowiadających elementom tablicy (macierzy) liczb całkowitych. Napisz program demonstrujący działanie tej metody.

Zadanie 28.14.

W klasie `TDouble` utwórz statyczną metodę `toInt()`, która będzie zwracać tablicę (macierz) o wartościach całkowitych, odpowiadających (zamiana przez rzutowanie) tablicy liczb zmiennoprzecinkowych podanej jako parametr. Napisz program demonstrujący działanie tej metody.

Zadanie 28.15.

W klasie `TInt` utwórz statyczną metodę `valueOf()`, która będzie budować tablicę (macierz) o wartościach całkowitych, odpowiadających (zamiana przez rzutowanie) tablicy liczb zmiennoprzecinkowych podanej jako parametr. Napisz program demonstrujący działanie tej metody.

Macierze, w których liczba wierszy jest równa liczbie kolumn, nazywamy *macierzami kwadratowymi*. Z macierzami kwadratowymi związany jest szereg pojęć, takich jak: *ślad macierzy*, *wyznacznik macierzy*, *macierz diagonalna*, *macierz trójkątna*, *macierz jednostkowa* i *macierz odwrotna*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.16.

Utwórz statyczną metodę `trace()` wyznaczającą ślad macierzy. Napisz program demonstrujący działanie tej metody.

Zadanie 28.17.

Utwórz statyczną metodę `getI()` zwracającą macierz jednostkową stopnia *n* (stopień podamy jako parametr wywołania metody). Napisz program demonstrujący działanie tej metody.

Zadanie 28.18.

Utwórz statyczną metodę `setI()` tworzącą z macierzy kwadratowej podanej jako parametr macierz jednostkową. Napisz program demonstrujący działanie tej metody.

Zadanie 28.19.

Utwórz metodę statyczną `det()` obliczającą wyznacznik macierzy kwadratowej. Do obliczenia wyznacznika użyj rozwinięcia Laplace'a. Napisz program demonstrujący działanie tej metody.

Zadanie 28.20.

Utwórz metodę statyczną `upperTriangular()` przekształcającą macierz kwadratową podaną jako parametr na macierz trójkątną górną. Napisz program demonstrujący działanie tej metody.

Zadanie 28.21.

Utwórz metodę statyczną `lowerTriangular()` przekształcającą macierz kwadratową podaną jako parametr na macierz trójkątną dolną. Napisz program demonstrujący działanie tej metody.

Zadanie 28.22.

Utwórz metodę statyczną `diagonal()` przekształcającą macierz kwadratową podaną jako parametr na macierz diagonalną. Napisz program demonstrujący działanie tej metody.

Zadanie 28.23.

Utwórz metodę statyczną `inverse()` obliczającą i zwracającą macierz odwrotną do macierzy kwadratowej podanej jako parametr. Napisz program demonstrujący działanie tej metody.

Zadanie 28.24.

Napisz program rozwiązujący układ *n*-równań liniowych z *n* niewiadomymi ($n < 10$). Układ równań rozwiąż, stosując metodę wyznaczników. Wszystkie niezbędne metody umieść w klasie programu.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.25.

Napisz program rozwiązujący układ *n*-równań liniowych z *n* niewiadomymi. Układ równań rozwiąż, stosując rachunek macierzy: $A \cdot X = B$, $X = A^{-1} \cdot B$, gdzie A — macierz podstawowa układu, X — wektor niewiadomych, B — kolumna wyrazów wolnych. Wszystkie niezbędne metody umieść w klasie programu.

Zadanie 28.26.

Napisz program rozwiązujący układ *n*-równań liniowych z *n* niewiadomymi. Układ równań rozwiąż, stosując metodę eliminacji. Wszystkie niezbędne metody umieść w klasie programu.

Zadanie 28.27.

Utwórz klasę `Matrix`, która na podstawie tablic dwuwymiarowych, podanych wymiarów macierzy lub innych obiektów klasy `Matrix` (konstruktor kopiujący) będzie umożliwiać tworzenie obiektów reprezentujących macierze. Utwórz metody ułatwiające dostęp do elementów macierzy (pól obiektu), wprowadzanie i wyświetlanie danych oraz wypełnianie macierzy wartościami losowymi. Napisz program demonstrujący działanie metod klasy `Matrix`.

Zadanie 28.28.

Dołącz do klasy `Matrix` metody umożliwiające wykonywanie podstawowych działań na macierzach: dodawanie, odejmowanie i mnożenie macierzy oraz mnożenie macierzy przez skalar. Napisz program demonstrujący działania na macierzach.

Zadanie 28.29.

Dołącz do klasy `Matrix` metody umożliwiające przekształcanie macierzy kwadratowych na postać trójkątną lub diagonalną. Napisz program demonstrujący działanie tych metod.

Zadanie 28.30.

Dołącz do klasy `Matrix` metodę umożliwiającą obliczanie wyznacznika macierzy. Napisz program demonstrujący działanie tej metody.

Zadanie 28.31.

Dołącz do klasy `Matrix` metodę umożliwiającą obliczanie macierzy odwrotnej. Napisz program demonstrujący obliczanie macierzy odwrotnej. Sprawdź uzyskany wynik, wykonując odpowiednie mnożenie.

Zadanie 28.32.

Wykorzystując możliwości klasy `Matrix` i obliczenia na macierzach, rozwiąż układ *n*-równań liniowych z *n* niewiadomymi.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.33.

Wykorzystując możliwości klasy `Matrix` i metodę wyznaczników, rozwiąż układ *n*-równań liniowych z *n* niewiadomymi.

> Utwórz metodę pomocniczą `replaceCol()`, która w macierzy wywołującej tę metodę zastąpi wskazaną kolumnę kolumną przekazaną jako parametr (będzie to kolumna wyrazów wolnych).

29. Obliczanie wartości funkcji, rekurencja i inne zadania

Zadanie 29.1.

Napisz aplikację testującą działanie podanej metody dla różnych argumentów. Określ, co oblicza ta metoda. Nadaj jej odpowiednią nazwę.

```
static double f(double x) {
    return (x > 0)?x:-x;
}
```

Zadanie 29.2.

Określono dwie funkcje: $\min(x, y) = \begin{cases} x, & \text{gdy } x \le y \\ y, & \text{gdy } x > y \end{cases}$ oraz $\max(x, y) = \begin{cases} x, & \text{gdy } x \ge y \\ y, & \text{gdy } x < y \end{cases}$.
Utwórz klasę `MinMax` z metodami `min()` i `max()` obliczającymi i zwracającymi wartości tych funkcji. Napisz aplikację pokazującą działanie tych metod.

Zadanie 29.3.

Określono funkcję $f(x) = \begin{cases} -1, & \text{gdy } x < 0 \\ 0, & \text{gdy } x = 0 \\ 1, & \text{gdy } x > 0 \end{cases}$. Co oblicza ta funkcja? Utwórz metodę obliczającą wartość tej funkcji. Napisz aplikację pokazującą działanie zbudowanej metody.

Zadanie 29.4.

Określono funkcję $f(x) = \begin{cases} 0, & gdy\ x = 0 \\ \dfrac{\lvert x \rvert}{x}, & gdy\ x \ne 0 \end{cases}$. Co oblicza ta funkcja? Utwórz metodę obliczającą wartość tej funkcji. Napisz aplikację pokazującą działanie zbudowanej metody.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 29.5.

Napisz aplikację testującą wartości funkcji określonych wzorami:

a) $f(x, y) = \dfrac{x + y + |x - y|}{2}$

b) $g(x, y) = \dfrac{x + y - |x - y|}{2}$

c) $h(x) = f(x, -x)$

Co obliczają te funkcje? Utworzonym metodom nadaj odpowiednie nazwy.

Zadanie 29.6.

Napisz definicje metod `square()` i `cube()` obliczających kwadrat i sześcian liczby `x`. Zastosuj utworzone metody do obliczenia kwadratów i sześcianów liczb:

a) całkowitych od *1* do *15*,

b) rzeczywistych od *1* do *3* z krokiem *0,25*.

Zadanie 29.7.

Potęgę o wykładniku całkowitym dodatnim określamy wzorem $a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n}$. Utwórz metodę `adoen()` obliczającą a^n. Napisz aplikację pokazującą działanie tej metody.

Zadanie 29.8.

Potęgę o wykładniku całkowitym nieujemnym możemy określić wzorem rekurencyjnym: $a^n = \begin{cases} 1, & \text{dla } n = 0 \\ a \cdot a^{n-1}, & \text{dla } n > 0 \end{cases}$. Napisz definicję metody rekurencyjnej `adoen()` obliczającej a^n oraz aplikację pokazującą działanie tej metody.

Zadanie 29.9.

Szybkie potęgowanie — inną wersję metody rekurencyjnej obliczającej a^n (szybszą ze względu na mniejszą liczbę wywołań rekurencyjnych i zredukowanie liczby mnożeń) możemy zrealizować na podstawie wzoru:

$$a^n = \begin{cases} 1 \quad dla \quad n = 0 \\ a \cdot \left(a^{\frac{n-1}{2}} \right)^2 & \text{dla } n \text{ nieparzystego } (n > 0) \\ \left(a^{\frac{n}{2}} \right)^2 & \text{dla } n \text{ parzystego } (n > 0) \end{cases}.$$

Ebookpoint.pl kopia dla: pawher@o2.pl

Utwórz metodę `adoen()` obliczającą a^n. Napisz aplikację pokazującą działanie tej metody.

Zadanie 29.10.

Potęgę o podstawie a różnej od 0 i wykładniku całkowitym n możemy zdefiniować w następujący sposób: $a^n = \begin{cases} \underbrace{a \cdot a \cdot \ldots \cdot a}_{n}, & \text{dla } n > 0 \\ 1, & \text{dla } n = 0 \\ \dfrac{1}{a^n}, & \text{dla } n < 0 \end{cases}$.

Napisz definicję metody `power()` (prototyp: `double power(double a, int n)`) obliczającej a^n dla dowolnej liczby całkowitej oraz aplikację pokazującą działanie tej funkcji.


Wskazówka

Funkcję `adoen` (zob. zadanie 29.7, 29.8 lub 29.9) obliczającą a^n dla $n > 1$ wykorzystamy w funkcji `power` (do obliczania a^n lub $a^{-n} = \dfrac{1}{a^n}$ dla $n = 1, 2, 3, \ldots$).

Zadanie 29.11.

Pierwiastek drugiego stopnia z liczby dodatniej a możemy obliczyć metodą iteracyjną na podstawie wzoru $x = \frac{1}{2}\left(x + \frac{a}{x}\right)$, przyjmując jako pierwsze przybliżenie $x = 1$. Obliczenia kontynuujemy do chwili, gdy różnica pomiędzy dwoma kolejnymi przybliżeniami pierwiastka będzie dostatecznie mała. Napisz metodę `sqrt()` (ang. *square root*) obliczającą pierwiastek kwadratowy z podanej liczby dodatniej. Zbuduj aplikację pokazującą działanie metody `sqrt()` i porównującą otrzymane wyniki z wynikami metody bibliotecznej `Math.sqrt()`.

Zadanie 29.12.

Pierwiastek trzeciego stopnia z liczby dodatniej a możemy obliczyć metodą iteracyjną na podstawie wzoru $x = \frac{1}{3}\left(2x + \frac{a}{x^2}\right)$, przyjmując jako pierwsze przybliżenie $x = 1$. Obliczenia kontynuujemy do chwili, gdy różnica pomiędzy dwoma kolejnymi przybliżeniami pierwiastka będzie dostatecznie mała. Napisz metodę `cbrt()` (ang. *cube root*) obliczającą pierwiastek trzeciego stopnia z podanej liczby dodatniej. Zbuduj aplikację pokazującą działanie metody `cbrt()` i porównującą otrzymane wyniki z wynikami metody bibliotecznej `Math.cbrt()`.


Uwaga

Podany wzór wynika z *metody Newtona-Raphsona* — iteracyjnego algorytmu wyznaczania przybliżonej wartości pierwiastka funkcji. Dotyczy to również wzoru z zadania 29.11.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 29.13.

Pierwiastek *n*-tego stopnia z liczby dodatniej *a* możemy obliczyć metodą iteracyjną na podstawie wzoru $x = \frac{1}{n}\left((n-1)x + \frac{a}{x^{n-1}}\right)$, przyjmując jako pierwsze przybliżenie $x = 1$. Obliczenia kontynuujemy do chwili, gdy różnica pomiędzy dwoma kolejnymi przybliżeniami pierwiastka będzie dostatecznie mała. Na podstawie podanego wzoru napisz metodę `nRoot()` obliczającą pierwiastek n-tego stopnia z podanej liczby dodatniej. Zbuduj aplikację pokazującą działanie metody `nRoot()` i porównującą otrzymane wyniki z wynikami uzyskanymi przy zastosowaniu funkcji bibliotecznej.


Uwaga

Ponieważ $\sqrt[n]{a} = a^{\frac{1}{n}}$, to wartość pierwiastka możemy obliczyć przy zastosowaniu funkcji `Math.pow(a, 1.0/n)`.


Wskazówka

Skorzystaj z opracowanej w zadaniu 29.7 metody `power()` do obliczania wartości x^{n-1}.

Zadanie 29.14.

W klasie `Math` zdefiniowano metody obliczające funkcje hiperboliczne — *sinus hiperboliczny* (`Math.sinh()`) i *cosinus hiperboliczny* (`Math.cosh()`). Napisz program wyświetlający na ekranie tablice wszystkich funkcji hiperbolicznych w przedziale $\langle -5, 5 \rangle$ z krokiem *0,1*. Wyniki obliczeń zapisz w pliku tekstowym *FunkcjeHiperboliczne.txt*.


Wskazówka

Wyniki pracy programu można zapisać w pliku tekstowym, stosując w konsoli polecenie:

```
java Z29_14 > FunkcjeHiperboliczne.txt
```

(przyjmujemy, że rozwiązanie zadania zapisano w pliku źródłowym *Z29_14.java*).

Zadanie 29.15.

W klasie `Math` zdefiniowano metodę `Math.exp()` obliczającą wartość funkcji wykładniczej e^x. Rozwiąż zadanie 29.14, nie korzystając z metod `Math.sinh()` i `Math.cosh()`.


Wskazówka

Wartości funkcji można wyliczyć na podstawie wzorów: $\sinh x = \frac{e^x - e^{-x}}{2}$, $\cosh x = \frac{e^x + e^{-x}}{2}$ itp.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 29.16.

Zbuduj klasę FH (funkcje hiperboliczne) zawierającą metody statyczne obliczające wartości wszystkich funkcji hiperbolicznych i funkcji do nich odwrotnych. Napisz aplikację pokazującą działanie tych metod.

Zadanie 29.17.

Zbuduj klasę FTD, która będzie zawierać metody statyczne obliczające wartość sześciu funkcji trygonometrycznych i sześciu funkcji do nich odwrotnych. Argumenty funkcji trygonometrycznych i wartości funkcji odwrotnych powinny być wyrażane w stopniach. Udostępnij również metody konwersji stopni na radiany i radianów na stopnie. Napisz aplikację pokazującą możliwości metod statycznych zawartych w tej klasie.

Zadanie 29.18.

Zbuduj klasę FTR, która będzie zawierać metody statyczne obliczające wartość sześciu funkcji trygonometrycznych i sześciu funkcji do nich odwrotnych. Argumenty funkcji trygonometrycznych i wartości funkcji odwrotnych powinny być wyrażane w radianach. Udostępnij również metody konwersji stopni na radiany i radianów na stopnie. Napisz aplikację pokazującą możliwości metod statycznych zawartych w tej klasie.

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 5.

Rozwiązania zadań

1. Historia Javy i pierwsze zadania

Zadanie 1.1.


Uwaga

> Podane rozwiązanie ma charakter poglądowy i było w całości aktualne, gdy powstawał ten tekst. Zmiany na stronach producenta Javy spowodowały, że Czytelnik nie może dokładnie wykorzystać tego rozwiązania i musi, wzorując się na nim, samodzielnie pobrać i zainstalować środowisko JDK.

Zakładamy, że dysponujesz komputerem klasy PC z systemem operacyjnym Windows XP lub nowszym i dostępem do internetu. Wykonaj następujące czynności:

1. Otwórz stronę internetową firmy Oracle *http://www.oracle.com/pl/index.html*.
2. Przejdź do działu *Do pobrania*.
3. Odszukaj link *Java SE* i przejdź do tej podstrony.
4. Wybierz wersję *Java 6 Update 27* i klikając przycisk *Download*, pobierz JDK (Java Development Kit) — *http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u27-download-440405.html*.
5. Zapoznaj się z licencją i zaakceptuj ją.
6. Wybierz plik instalacyjny właściwy dla posiadanego systemu operacyjnego i zapisz go na swoim dysku lokalnym:

   jdk-6u27-windows-i586.exe (76,81 MB) — dla systemu 32-bitowego,

   jdk-6u27-windows-x64.exe (67,40 MB) — dla systemu 64-bitowego.
7. Otwórz zapisany plik instalacyjny i przeprowadź proces instalacji.

Ebookpoint.pl kopia dla: pawher@o2.pl

Podczas typowej instalacji środowisko JDK zostanie zainstalowane w folderze *C:\Program Files\Java\jdk1.6.0_27\bin*[1]. Pliki binarne (wykonywalne) znajdują się w podfolderze *\bin*. Należy dodać do systemowej ścieżki przeszukiwań ścieżkę *C:\Program Files\Java\jdk1.6.0_27\bin* (lub inną, wynikającą z przebiegu instalacji). W *Panelu sterowania* otwórz aplet *System* i przejdź do zakładki *Zaawansowane*. Kliknij przycisk *Zmienne środowiskowe*, na liście *Zmienne użytkownika* odszukaj i edytuj zmienną *Path* (dopisz potrzebną ścieżkę).

Zadanie 1.2.

Otwórz konsolę (w Windows: *Start/Uruchom...* — w polu edycyjnym w pisz polecenie `cmd` i kliknij przycisk *OK*), wpisz w konsoli polecenie `java` i naciśnij *Enter*. Jeśli w konsoli zostanie wyświetlona informacja o sposobie uruchomiania programu (`help`), to znaczy, że masz zainstalowane środowisko uruchomieniowe Javy (JRE). Po wpisaniu polecenia `java -version` uzyskasz informację o wersji posiadanej maszyny wirtualnej Javy.

Wpisz w konsoli polecenie `javac`. Jeśli w konsoli pojawi się informacja o sposobie i opcjach uruchomienia kompilatora, to masz poprawnie zainstalowane środowisko JDK. Możesz sprawdzić wersję kompilatora (`javac -version`). Niepowodzenie testu, w postaci komunikatu: `Nazwa 'javac' nie jest rozpoznawana jako polecenie wewnętrzne lub zewnętrzne, program wykonywalny lub plik wsadowy`, nie świadczy o braku JDK. Być może nie została poprawnie ustawiona ścieżka dostępu (`path`) do folderu `bin` zawierającego kompilator *javac.exe*. Sprawdź to.

Zadanie 1.3.

Uruchom aplikację w linii komend i przekieruj wyświetlane dane do pliku, np. `java -? > java.txt` lub `java -help > java.txt` (wystarczy również polecenie `java > java.txt`). Następnie w dowolnym edytorze tekstowym sformatuj ten tekst według swoich potrzeb i wydrukuj. Możesz przetłumaczyć opisy (ale nie słowa będące opcjami!) na język polski, np.:

```
-? -help      print this help message
-? -help      wyświetl komunikat pomocy
```

Wydrukuj „ściągawkę", często z niej zapewne skorzystasz.

```
Usage: java [-options] class [args...] (to execute a class)
   or  java [-options] -jar jarfile [args...] (to execute a jar file)
where options include:
-client to select the "client" VM
-server to select the "server" VM
-hotspot is a synonym for the "client" VM [deprecated]. The default VM is client.
-cp <class search path of directories and zip/jar files>
-classpath <class search path of directories and zip/jar files>
           A ; separated list of directories, JAR archives,
           and ZIP archives to search for class files.
```

[1] Wersja 32-bitowa JDK w 64-bitowym systemie operacyjnym Windows 7 zainstaluje się w folderze *C:\Program Files (x86)\Java\jdk1.6.0_27\bin*.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
-D<name>=<value>  set a system property
-verbose[:class|gc|jni] enable verbose output
-version print product version and exit
-version:<value>  require the specified version to run
-showversion  print product version and continue
-jre-restrict-search | -jre-no-restrict-search include/exclude user private JREs
 in the version search
-? -help print this help message
-X print help on non-standard options
-ea[:<packagename>...|:<classname>]
-enableassertions[:<packagename>...|:<classname>] enable assertions
-da[:<packagename>...|:<classname>]
-disableassertions[:<packagename>...|:<classname>]  disable assertions
-esa | -enablesystemassertions  enable system assertions
-dsa | -disablesystemassertions  disable system assertions
-agentlib:<libname>[=<options>] load native agent library <libname>, e.g.
-agentlib:hprof, see also, -agentlib:jdwp=help and -agentlib:hprof=help
-agentpath:<pathname>[=<options>]  load native agent library by full pathname
-javaagent:<jarpath>[=<options>]  load Java programming language agent, see
 java.lang.instrument
-splash:<imagepath>  show splash screen with specified image
```

Zadanie 1.4.

Uruchomienie aplikacji *javac.exe* bez podania parametrów powoduje wyświetlenie następującej instrukcji:

```
Usage: javac <options> <source files>
where possible options include:
-g                        Generate all debugging info
-g:none                   Generate no debugging info
-g:{lines,vars,source} Generate only some debugging info
-nowarn                   Generate no warnings
-verbose                  Output messages about what the compiler is doing
-deprecation              Output source locations where deprecated APIs are used
-classpath <path>         Specify where to find user class files and annotation
 processors
-cp <path>                Specify where to find user class files and annotation
 processors
-sourcepath <path>        Specify where to find input source files
-bootclasspath <path>  Override location of bootstrap class files
-extdirs <dirs>           Override location of installed extensions
-endorseddirs <dirs>   Override location of endorsed standards path
-proc:{none,only}         Control whether annotation processing and/or compilation
 is done.
-processor <class1>[,<class2>,<class3>...] Names of the annotation processors
 to run; bypasses default discovery process
-processorpath <path>  Specify where to find annotation processors
-d <directory>             Specify where to place generated class files
-s <directory>             Specify where to place generated source files
-implicit:{none,class} Specify whether or not to generate class files for
 implicitly referenced files
-encoding <encoding>   Specify character encoding used by source files
-source <release>         Provide source compatibility with specified release
-target <release>         Generate class files for specific VM version
-version                  Version information
-help                     Print a synopsis of standard options
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
-Akey[=value]          Options to pass to annotation processors
-X                     Print a synopsis of nonstandard options
-J<flag>               Pass <flag> directly to the runtime system
```

Przechwycenie tego tekstu poleceniem `javac > opis.txt` do pliku *opis.txt* się nie powiedzie. Możemy natomiast tekst w konsoli zaznaczyć (menu wywołane prawym przyciskiem myszy) i skopiować zaznaczony fragment lub cały tekst do schowka (naciskając klawisz *Enter*).

Zadanie 1.5.

1. Plik *console.bat* — otwieranie konsoli w bieżącym folderze.

   ```
   cmd
   ```

   Możesz ten plik wsadowy rozbudować tak, aby automatycznie wywoływał kolejne dwa pliki wsadowe.

   ```
   call setpath
   call setdrive
   cls
   cmd
   ```

   Do pliku *console.bat* utwórz skrót na pulpicie lub w innym dogodnym miejscu.

2. Plik *setpath.bat* — ustawianie ścieżki dostępu do folderu *bin* na komputerach, na których nie mamy uprawnień do zmian konfiguracyjnych w systemie lub nie chcemy tych zmian dokonywać[2].

   ```
   path C:\Program Files\Java\jdk1.6.0_27\bin; %PATH%
   ```

3. Plik *setdrive.bat* — ustawienie folderu z plikami źródłowymi jako dysku *X:*[3].

   ```
   subst X: H:\!JAVA_ZZ\Pliki
   X:
   ```

4. Plik *kompiluj.bat* — kompilacja pliku źródłowego o nazwie podanej jako parametr dla pliku wsadowego (nazwę podamy bez rozszerzenia *.java* i to jest nasze małe uproszczenie pracy).

   ```
   javac %1.java
   ```

   Możemy ten plik rozbudować, dodając ścieżkę do kompilatora.

   ```
   C:\Program Files\Java\jdk1.6.0_27\bin\javac %1.java
   ```

5. Plik *uruchom.bat* — uruchomienie wirtualnej maszyny Javy (JVM) i przekazanie do wykonania kodu skompilowanej klasy. Nazwę klasy podajemy jako parametr pliku wsadowego (tu żadnego uproszczenia pracy nie widać).

   ```
   java %1
   ```

[2] Na komputerze Czytelnika ścieżka może być inna, zależna od tego, gdzie zainstalowano JDK. Autor używa ścieżki dla standardowego przebiegu procesu instalacji środowiska Javy.

[3] Autor w czasie opracowywania niniejszego zbioru zadań korzystał z pendrive'a, który instalował się w systemie jako dysk *H:*. Na materiały związane z tworzonym zbiorem autor utworzył folder *!JAVA_ZZ*, a pliki źródłowe zapisywał w folderze *!JAVA_ZZ\Pliki*. Na komputerze Czytelnika ścieżka z plikami może być inna.

Ebookpoint.pl kopia dla: pawher@o2.pl

Program *cmd.exe* umożliwia użycie dziesięciu zmiennych do oznaczania parametrów pliku wsadowego (od `%0` do `%9`). Zmienna `%0` przekazuje nazwę pliku wsadowego, a zmienne od `%1` do `%9` zawierają kolejne argumenty wpisane przez użytkownika w wierszu polecenia. Możemy zatem do pliku wsadowego przekazać nazwę klasy (`%1`) i maksymalnie 8 parametrów (od `%2` do `%9`).

```
java %1 %2 %3 %4 %5 %6 %7 %8 %9
```

6. Plik *edytuj.bat* — uruchomienie systemowego Notatnika (*notepad.exe*) i przekazanie do edycji kodu źródłowego klasy. Nazwę klasy podajemy jako parametr pliku wsadowego (nazwa pliku powstanie z nazwy klasy i rozszerzenia *.java*).

```
start notepad.exe %1.java
```

Przypomnijmy na koniec, że dysk *X:* możemy usunąć poleceniem `subst X: /D`. Nie powoduje to oczywiście utraty danych, które są bezpiecznie przechowywane w folderze dotychczas skojarzonym z napędem *X:*. Konsolę natomiast zamkniemy poleceniem `exit`. Na listingu 1.1 przedstawiono wersję pliku *console.bat* używaną przez autora.

Listing 1.1. *console.bat*

```
path C:\Program Files\Java\jdk1.6.0_27\bin; %PATH%
subst X: H:\!JAVA_ZZ\Pliki
X:
cls
call cmd
c:
subst X: /D
```

Dwa końcowe wiersze są wykonywane dopiero po zamknięciu konsoli poleceniem `exit` (nie należy zamykać okna konsoli w inny sposób).

Na rysunku 1.1 widoczne jest rozwiązanie zastosowane w konfiguracji skrótu na pulpicie.

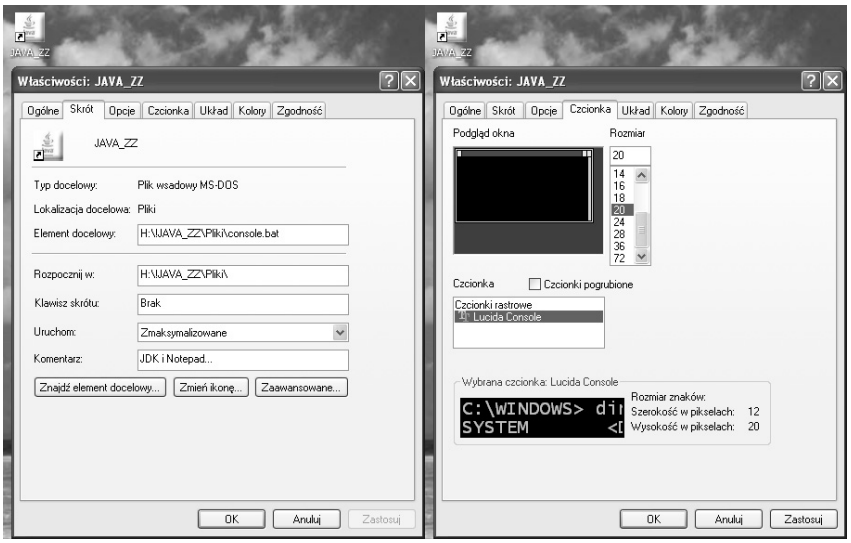
2. JDK, Notatnik i klasyczny przykład „Hello World”

Zadanie 2.1. Hello.java, Z02_1.java

Kolejność postępowania będzie następująca:

1. Przygotuj do pracy dysk *X:* (otwórz skrót z pulpitu). Dalej pracuj w konsoli na dysku *X:*.
2. Poleceniem `edytuj Hello` utwórz plik tekstowy *Hello.java* (w Notatniku) i przepisz podany kod. Zapisz tekst (*Ctrl+S*) i zamknij Notatnik (*Alt+F4*).

Ebookpoint.pl kopia dla: pawher@o2.pl



Rysunek 1.1. *Propozycja konfiguracji skrótu na pulpicie*

```
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

3. Skompiluj kod źródłowy poleceniem `kompiluj Hello` (stosując plik wsadowy *kompiluj.bat*) lub `javac Hello.java`. Jeśli kompilacja się nie powiodła (są komunikaty o błędach), wróć do punktu 2.

4. Uruchom aplikację poleceniem `uruchom Hello` (plik wsadowy *uruchom.bat*) lub `java Hello`. Pamiętaj o wielkości liter (Java rozróżnia wielkość liter — słowa: `hello`, `Hello` i `HELLO` są trzema różnymi identyfikatorami). Jako parametr uruchomienia podajemy nazwę klasy, a nie nazwę pliku.

Pierwsza aplikacja została uruchomiona. Przedstawioną tu procedurę stosuj do dalszych zadań. Zapamiętaj również kilka ważnych informacji.

Prosta aplikacja w języku Java składa się z jednej publicznej klasy. Nazwę klasy rozpoczynamy zwykle od wielkiej litery. Jeśli nazwa klasy składa się z kilku słów, to łączymy je w jedno słowo, stosując tzw. notację wielbłądzią (*CamelCase*), np.: `MojaPierwszaKlasa`.

Klasa ta musi zawierać publiczną i statyczną metodę `main`, od której rozpoczyna się wykonanie aplikacji.

Nazwa pliku źródłowego musi być zgodna (również pod względem wielkości liter) z nazwą klasy. Wymagane jest rozszerzenie *.java*, np. *MojaPierwszaKlasa.java*.

Kompilujemy kod źródłowy klasy, np. `javac MojaPierwszaKlasa.java`. Efektem kompilacji jest kod bajtowy (*B-kod*, *betakod*, ang. *bytecode*) zapisany w pliku *MojaPierwszaKlasa.class*.

Aplikację uruchamiamy, otwierając wirtualną maszynę Javy (JVM) i przekazując do interpretacji (jako parametr) nazwę skompilowanej klasy, np. `java MojaPierwszaKlasa`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Umiesz już przepisać kod, skompilować plik źródłowy i uruchomić aplikację. Przeanalizujmy ten prosty przykład.

Zacznijmy od budowy klasy. Kod klasy składa się ze słowa kluczowego `class`, identyfikatora (nazwy klasy) i ciała klasy ujętej w nawiasy klamrowe {}, np. `class Pusta { }`. W naszym przypadku występuje jeszcze słowo kluczowe `public`, określające dostęp (publiczny) do klasy.

W ciele klasy występuje publiczna i statyczna metoda `main`:

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

Słowo kluczowe `public` oznacza, że metoda `main()` jest widoczna dla wszystkich elementów programu. Metoda statyczna (słowo kluczowe `static`) jest dostępna bez konieczności tworzenia obiektu klasy. Przed nazwą metody `main` występuje słowo kluczowe `void`, informujące, że metoda nie zwraca żadnej wartości (pusty typ wyniku). W nagłówku metody `main()` konieczne jest zadeklarowanie tablicy argumentów `String args[]` lub `String[] args` (obie postacie deklaracji oznaczają to samo). `String` jest nazwą klasy (typu obiektowego) służącej do wykonywania operacji na łańcuchach znaków (tekstach), `args` jest identyfikatorem deklarowanego obiektu[4], a symbol `[]` oznacza tablicę złożoną z kilku obiektów. Podczas uruchomienia aplikacji ta tablica jest wypełniana przez system parametrami wywołania. Szczegóły wkrótce. W tym przykładzie nie korzystamy z tablicy argumentów, ale taka deklaracja musi być w nagłówku metody `main()`.

W ciele metody występuje jeden wiersz, odpowiedzialny za wyświetlenie tekstu *Hello World* w konsoli:

```
System.out.println("Hello World");
```

W klasie `System` zdefiniowana jest statycznie zmienna obiektowa `out`. Metody tej zmiennej działają na standardowym strumieniu wyjściowym. Użyta w przykładzie metoda `println()` wysyła na standardowe wyjście (konsolę) dane przekazane jako parametr podczas wywołania metody i kod końca wiersza[5]. W tym przykładzie argumentem jest stała wartość tekstowa, czyli *literał łańcuchowy* `"Hello World"`.

Zadanie 2.2. Witam.java, Z02_2.java

To kolejny przykład wykorzystania metody `System.out.println` opisanej w rozwiązaniu zadania 2.1. Problem prawidłowego wyświetlania liter z polskimi znakami diakrytycznymi w konsoli omówiono, podając wskazówkę do zadania.

[4] Ten identyfikator można zmienić, ale nie ma potrzeby tego robić. Słowo `args` jest po prostu skrótem od słowa *argumenty* (ang. ***arg**uments*).

[5] Parametr może być wielkością stałą (*literałem*), zmienną dowolnego typu lub wyrażeniem. Metoda `println()` wywołana bez parametru powoduje tylko przeniesienie kursora na początek nowej linii. Istnieje też podobnie działająca metoda `print()`, która wypisuje informację na standardowym wyjściu, ale nie przenosi kursora do nowego wiersza.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Witam {
    public static void main(String args[]) {
        System.out.println("Witaj, świecie.");
        System.out.println("Uczę się programować w języku Java.");
    }
}
```

Zadanie 2.3. Adres.java, Z02_3.java

W łańcuchu znaków umieszczono specjalny znak `\n`, powodujący przejście tekstu umieszczonego za znakiem do nowego wiersza.

```
public class Adres {
    public static void main(String args[]) {
        /* Dane osoby i adres są zupełnie fikcyjne */
        System.out.println("Ewa Nowak\nul. Krótka 1\n11-111 Fikcja
            Dolna");
    }
}
```

W ten sposób w łańcuchu możemy umieścić inne znaki specjalne: `\b` — backspace, `\t` — tabulacja, `\r` — powrót karetki, `\"` — cudzysłów, `\'` — apostrof, `\\` — lewy ukośnik.

Zadanie 2.4. Etykieta.java, Z02_4.java

```
public class Etykieta {
    public static void main(String args[]) {
        System.out.println("******* Programowanie *******");
        System.out.println("*  obiektowe w języku Java  *");
        System.out.println("*         Jan Nowak         *");
        System.out.println("*****************************");
    }
}
```

3. Znaki, tablice znaków i klasa String

Zadanie 3.1. Witaj.java, Z03_1.java

Deklarujemy tablicę znaków (`char[]`) o nazwie `witaj`. Rozmiar tablicy nie został ustalony. Ponieważ łączymy deklarację z inicjowaniem elementów tablicy, to kompilator na tej podstawie obliczy rozmiar tablicy i wypełni ją odpowiednimi wartościami. Zwróćmy uwagę na sposób zapisu znaków z użyciem apostrofów i zapamiętajmy zasadniczą różnicę: `'A'` — znak, litera *A*; `"A"` — łańcuch znaków (jednoelementowy), zawierający jeden znak *A*. Jeśli potrzebujemy wyświetlić wszystkie znaki zapisane w tablicy (w kolejności ich występowania), to możemy użyć metody `print` lub `println`, przekazując nazwę tablicy jako parametr wywołania metody.

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

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        char[] witaj = {'D', 'z', 'i', 'e', 'ń', ' ', 'd', 'o',
            'b', 'r', 'y'};
        System.out.println(witaj);
    }
}
```

Zadanie 3.2. Informatyka.java, Z03_2.java

W działaniach związanych z przetwarzaniem tablic często wykorzystujemy instrukcje iteracyjne (pętle). Zawartość tablicy możemy wyświetlić, używając pętli typu `for each` (wprowadzonej do języka Java SE 5.0):

```
for(typ_zmiennej zmienna : kolekcja) instrukcja;
```

W powyższym zapisie `for` jest słowem kluczowym definiującym pętlę, identyfikator *`zmienna`* oznacza zmienną typu `typ_zmiennej` zawartego w kolekcji, dwukropek (`:`) oddziela zmienną od identyfikatora kolekcji (wymaga tego składnia pętli `for each`), *`kolekcja`* w tym przypadku oznacza nazwę tablicy[6]. Identyfikator zmiennej i nazwa kolekcji ujęte są w nawiasy `( )`, natomiast za nawiasem znajduje się *`instrukcja`* do powtarzania. Instrukcja powtarzana jest tyle razy, ile elementów zawiera kolekcja; w każdym powtórzeniu cyklu wartość zmiennej jest równa wartości kolejnego elementu kolekcji. Należy dodać, że pętla typu `for each` służy do odczytywania kolekcji i za pośrednictwem zmiennej nie możemy modyfikować wartości elementów kolekcji.

Pętla `for(char x : a) System.out.print(x);` wyświetli w konsoli po kolei wszystkie znaki zapisane w tablicy `a`. Oczywiście możemy z wartością zmiennej `x` wykonać inne, bardziej skomplikowane operacje niż tylko wyświetlanie w konsoli.

a) W pętli `for(char znak : dane) ...` pobieramy do zmiennej `znak` kolejne elementy z tablicy `dane`. Każdy pobrany z tablicy `znak` wyświetlamy w odrębnym wierszu konsoli `System.out.println(znak)`.

b) Pisanie *tekstem rozstrzelonym* (*spacjowanie*) polega na pisaniu tekstu znak po znaku, oddzielając kolejne znaki odstępami (spacjami). W pętli pobieramy z tablicy kolejne znaki i wyświetlamy w konsoli `System.out.print(znak); System.out.print(" ");`.

c) W klasie `Character` zdefiniowano statyczną metodę `toUpperCase()` zmieniającą małą literę na odpowiadającą jej wielką literę. Wykorzystujemy tę metodę do zamiany znaków odczytywanych z tablicy `System.out.print(Character.toUpperCase(znak));`.

d) Pobrane z tablicy `dane` znaki (małe lub wielkie litery), zamieniamy na małe litery (`Character.toLowerCase(znak)`) i wyświetlamy w konsoli.

```
public class Informatyka {
    public static void main(String args[]) {
        char dane[] = {'I', 'n', 'f', 'o', 'r', 'm', 'a', 't', 'y',
            'k', 'a'};
        /* Zadanie 3.2a */
        System.out.println("Pionowo: ");
```

[6] Dalej będą przedstawione inne kolekcje.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(char znak : dane)
                System.out.println(znak);
            System.out.println();
            /* Zadanie 3.2b */
            System.out.print("Tekst rozstrzelony: ");
            for(char znak : dane) {
                System.out.print(znak);
                System.out.print(" ");
            }
            System.out.println();
            /* Zadanie 3.2c */
            System.out.print("Wielkie litery: ");
            for(char znak : dane)
                System.out.print(Character.toUpperCase(znak));
            System.out.println();
            /* Zadanie 3.2d */
            System.out.print("Małe litery: ");
            for(char znak : dane)
                System.out.print(Character.toLowerCase(znak));
            System.out.println();
        }
    }
```


Uwaga

Metody `toLowerCase()` i `toUpperCase()` zmieniają wyłącznie wielkość liter, nie powodują żadnych zmian w pozostałych znakach.

Zadanie 3.3. Programowanie.java, Z03_3.java

Stosując podstawienie `dane[0] = Character.toUpperCase(dane[0]);`, zmienimy pierwszy znak zapisany w tablicy `dane`. Podobnie można postąpić z pozostałymi znakami. Indeksy znaków zapisanych w tablicy zmieniają się w zakresie od *0* do wartości o *1* mniejszej od rozmiaru tablicy. Dostęp do wszystkich znaków w tablicy `dane` uzyskamy, stosując pętlę typu `for`:

```
for(int i=0; i < dane.length; ++i) {
// Tu wykonaj przekształcenie związane z elementem dane[i].
}
```

Pętla `for` będzie dalej omówiona dokładniej; na razie zapamiętaj, że zmienna `i` typu całkowitego `int` będzie w tym przypadku zmieniała się od *0* (na początku inicjujemy zmienną `int i=0;`) do wartości `dane.length-1`. Instrukcja zawarta w ciele pętli wykonuje się, gdy spełniony jest warunek `i < dane.length`. Po każdym wykonaniu pętli wartość zmiennej `i` jest zwiększana o `1` (operator inkrementacji `++i`[7]). Pętla wykona się dokładnie tyle razy, ile elementów ma tablica `dane`, a zmienna `i` przyjmie kolejno wartości indeksów wszystkich elementów tablicy.

Słowo kluczowe `int` jest nazwą typu do przechowywania liczb całkowitych w zakresie od -2^{31} do $2^{31}-1$. Konstrukcja `int a;` jest deklaracją zmiennej `a` typu `int`, natomiast `int a = wartość;` deklaruje zmienną i inicjuje jej wartość początkową (`wartość` może być *literałem* lub *wyrażeniem*).

[7] Zapis `++i` jest równoważny podstawieniu `i = i+1`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Programowanie {
    public static void main(String args[]) {
        char dane[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', 'o', 'w',
            'a', 'n', 'i', 'e'};
        System.out.print("Tablica znaków: ");
        System.out.println(dane);
        /* Zmiana pierwszej litery na wielką */
        dane[0] = Character.toUpperCase(dane[0]);
        System.out.print("Pierwsza litera wielka: ");
        System.out.println(dane);
        /* Zmiana wszystkich liter na wielkie */
        for(int i=0; i < dane.length; ++i)
            dane[i] = Character.toUpperCase(dane[i]);
        System.out.print("Wszystkie litery wielkie: ");
        System.out.println(dane);
    }
}
```

Zadanie 3.4. Wspak1.java, Z03_4.java

Do przeglądania tablicy zastosujemy pętlę `for`, w której następuje odliczanie w dół (od wartości większych do mniejszych:

```
for(int i=dane.length-1; i >= 0; --i) {
// Tu wykonamy przekształcenie związane z elementem dane[i].
}
```

Na początku zmienną `i` inicjujemy wartością równą indeksowi ostatniego elementu (rozmiar tablicy zmniejszony o *1* — `int i=dane.length-1`). Instrukcje są wykonywane, gdy `i >= 0` (`i` jest większe lub równe `0`). Po każdym wykonaniu instrukcji wartość zmiennej `i` jest zmniejszana o *1* (dekrementacja `--i`[8]).

```
public class Wspak1 {
    public static void main(String args[]) {
        char dane[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', 'o', 'w',
            'a', 'n', 'i', 'e'};
        System.out.print("Normalna kolejność znaków: ");
        System.out.println(dane);
        System.out.print("Odwrotna kolejność znaków: ");
        for(int i = dane.length-1; i >= 0; --i)
            System.out.print(dane[i]);
        System.out.println();
    }
}
```

Zadanie 3.5. Wspak2.java, Z03_5.java

Do zamiany wartości dwóch elementów użyjemy zmiennej pomocniczej według schematu: `p = a; a = b; b = p;` — spowoduje to wymianę wartości pomiędzy zmiennymi `a` i `b`. Odwracanie tablicy znaków zrealizujemy, zamieniając pierwszy znak z ostatnim, drugi z przedostatnim — i tak do chwili, gdy dojdziemy do środka tablicy. W tym celu wykorzystamy następujące zmienne: `i` — indeks pierwszego znaku (indeks ten zwiększa

[8] Zapis `--i` jest równoważny podstawieniu `i = i-1`.

Ebookpoint.pl kopia dla: pawher@o2.pl

się, przechodząc od początku (`int i = 0`) do środka tablicy), `j` — indeks ostatniego znaku (indeks ten zmniejsza się, przechodząc od końca (`int j = dane.length-1`) do środka tablicy), `tmp` — zmienna pomocnicza do zamiany wartości dwóch elementów tablicy. Czynności zamiany elementów są powtarzane, gdy `i < j`. Na koniec każdego cyklu zwiększamy indeks `i` (`++i`) i zmniejszamy indeks `j` (`--j`), zbliżając się do środka tablicy.

```
public class Wspak2 {
    public static void main(String args[]) {
        char dane[] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', 'o',
           'w', 'a', 'n', 'i', 'e'};
        System.out.print("Normalna kolejność znaków: ");
        System.out.println(dane);
        for(int i = 0, j = dane.length-1; i < j ; ++i, --j) {
            char tmp = dane[i];
            dane[i] = dane[j];
            dane[j] = tmp;
        }
        System.out.print("Odwrotna kolejność znaków: ");
        System.out.println(dane);
    }
}
```

Zadanie 3.6. DemoCharacter.java, Z03_6.java

Pierwszy wiersz kodu `import static java.lang.System.*;` powoduje zaimportowanie do naszej klasy wszystkich elementów statycznych z klasy `java.lang.System` (w tym obiektu `out`). W dalszej części kodu możemy posługiwać się tym obiektem bez podawania nazwy klasy, z której pochodzi. Zamiast odwołania `System.out.println(...);` wystarczy `out.println(...);`.

Na początku umieszczono kilka informacji o metodzie `digit` (ang. *digit* — cyfra):

- `out.println("Klasa: java.lang.Character");` — klasa `Character` znajduje się w pakiecie `java.lang`.
- `out.println("Metoda statyczna: digit\n");` — omawiana metoda jest metodą statyczną, nie potrzebujemy tworzyć obiektu klasy `Character`, aby skorzystać z tej metody.
- `out.println("static int digit(int ch, int radix)");` — nagłówek metody `digit` zawiera: słowo kluczowe `static` (metoda jest statyczna), typ zwracanej wartości `int` i listę parametrów wywołania metody `(int ch, int radix)`.
- `out.println("Returns the numeric value of the character ch in the specified radix.");` — pierwsze zdanie z dokumentacji metody, informujące o działaniu metody. Metoda `digit` zwraca liczbę będącą wartością cyfry reprezentowanej przez znak `ch` w układzie pozycyjnym o określonej podstawie (`radix`).

Następnie zadeklarowano przykładową tablicę znaków zawierającą znaki napisu *Euro 2012* — `char znak[] = {'E', 'u', 'r', 'o', ' ', '2', '0', '1', '2'};`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Działanie metody pokazano w dwóch przypadkach, obliczając wartości znaków w tablicy jako cyfr układów dziesiątkowego (`radix = 10`) oraz szesnastkowego (`radix = 16`).

W pętli `for` pobieramy kolejne znaki z tablicy, wyświetlamy w konsoli ich postać znakową i wartość jako cyfry w określonym układzie liczbowym. Sprawdzamy cyfry układu dziesiątkowego:

```
for(char z : znak)
    out.println("Znak: "+z+"  Cyfra: "+Character.digit(z, 10));
```

Wynik `-1` oznacza, że znak nie jest cyfrą w tym układzie liczbowym.

```
Wartość znaku jako cyfry w układzie dziesiątkowym (radix = 10)
Znak: E  Cyfra: -1
Znak: u  Cyfra: -1
Znak: r  Cyfra: -1
Znak: o  Cyfra: -1
Znak:    Cyfra: -1
Znak: 2  Cyfra: 2
Znak: 0  Cyfra: 0
Znak: 1  Cyfra: 1
Znak: 2  Cyfra: 2
```

Podobnie sprawdzimy cyfry w układzie szesnastkowym:

```
for(char z : znak)
    out.println("Znak: "+z+"  Cyfra: "+Character.digit(z, 16));
```

Po wykonaniu kodu zauważymy, że litera *E* jest cyfrą w układzie szesnastkowym (wartość dziesiętna *14*).

```
Wartość znaku jako cyfry w układzie szesnastkowym (radix = 16)
Znak: E  Cyfra: 14
Znak: u  Cyfra: -1
Znak: r  Cyfra: -1
Znak: o  Cyfra: -1
Znak:    Cyfra: -1
Znak: 2  Cyfra: 2
Znak: 0  Cyfra: 0
Znak: 1  Cyfra: 1
Znak: 2  Cyfra: 2
```

Zadanie 3.7. Cyfry.java, Z03_7.java

Zgodnie z podaną wskazówką odrzucamy najprostsze rozwiązanie: `char[] cyfry = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};`.

Wykorzystamy fakt, że kody cyfr dziesiętnych (*0123456789*) są liczbami z zakresu od *48* do *57*. Aby zamienić kod znaku (liczbę całkowitą) na odpowiadający mu znak, posłużymy się rzutowaniem typów, np.: `(char) 48` — liczbę całkowitą *48* rzutujemy na typ `char` i otrzymujemy znak `'0'`.

```
public class Cyfry {
    public static void main(String args[]) {
        char[] cyfry = new char[10];
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        for(int i = 0; i < 10; ++i)
            cyfry[i] = (char)(i+48);
        System.out.print("Cyfry układu dziesiątkowego: ");
        System.out.println(cyfry);
    }
}
```

Pętlę wypełniającą tablicę możemy również zapisać w postaci:

```
for(int i = 48; i < 58; ++i)
    cyfry[i] = (char)i;
```

Zadanie 3.8. Cyfry16.java, Z03_8.java

Zadanie to możemy rozwiązać podobnie jak zadanie 3.7. Do zapisania liczby w układzie szesnastkowym potrzebujemy szesnastu cyfr — dziesięć z nich to cyfry znane z układu dziesiątkowego (*0123456789* — znaki o kodach od *48* do *57*), jako sześć pozostałych cyfr wykorzystamy litery *ABCDEF* (znaki o kodach od *65* do *70*).

```
public class Cyfry16 {
    public static void main(String args[]) {
        char[] cyfry = new char[16];
        for(int i = 0; i < 10; ++i)
            cyfry[i] = (char)(i+48);
        for(int i = 10; i < 16; ++i)
            cyfry[i] = (char)(i+55);
        System.out.print("Cyfry układu szesnastkowego: ");
        System.out.println(cyfry);
    }
}
```

Do rozwiązania zadania możemy wykorzystać statyczną metodę `forDigit` (z dwoma parametrami `int digit` i `int radix`) z klasy `Character`, zamieniającą wartość liczbową (parametr `digit`) na odpowiadającą jej cyfrę (znak) w układzie liczbowym o podstawie `radix`.

```
public class Cyfry16 {
    public static void main(String args[]) {
        char[] cyfry = new char[16];
        for(int i = 0; i < 16; ++i)
            cyfry[i] = Character.forDigit(i, 16);
        System.out.print("Cyfry układu szesnastkowego: ");
        System.out.println(cyfry);
    }
}
```

Metoda `forDigit` zwróci cyfry o wartości większej od *9* jako małe litery (*abcdef*). Możemy przyjąć takie rozwiązanie lub dodatkowo zamienić te znaki na wielkie litery (stosując metodę poznaną w rozwiązaniu zadania 3.2):

```
for(int i = 0; i < 16; ++i)
    cyfry[i] = Character.toUpperCase(Character.forDigit(i, 16));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

4. Klasa String — operacje na tekstach

Zadanie 4.1. TestString.java, Z04_1.java

`"Zadania z programowania."` — łańcuch znaków stanowiący punkt wyjścia do dalszych przekształceń.

`"Zadania z programowania.".charAt(0)` — wyodrębnienie znaku o indeksie `0` (pierwszy znak łańcucha).

`"Zadania z programowania.".length()` — długość łańcucha (liczba znaków, w tym przypadku `24`).

`"Zadania z programowania.".charAt(23)` — wyodrębnienie znaku o indeksie `23` (ostatni znak łańcucha).

`"Zadania z programowania.".toUpperCase()` — zamiana liter w łańcuchu na wielkie litery (oczywiście zwracana jest kopia łańcucha, na której dokonano zmian, oryginał pozostaje nienaruszony).

`"Zadania z programowania.".toLowerCase()` — zamiana liter w łańcuchu na małe litery (jw.).

`"Zadania z programowania.".indexOf('z')` — indeks pozycji pierwszego wystąpienia znaku `z`. Jeśli podany znak nie występuje w łańcuchu, to zwracana jest wartość `-1`.

`"Zadania z programowania.".indexOf("prog")` — indeks pozycji pierwszego wystąpienia ciągu `"prog"`. Jeśli podany ciąg nie występuje w łańcuchu, to zwracana jest wartość `-1`.

`"Zadania z programowania.".replace('.', '?')` — zwraca nowy ciąg, w którym wszystkie znaki kropki (`.`) zostaną zastąpione znakiem pytajnika (`?`).

`"Zadania z programowania.".replace("adania", "dania")` — zastąpienie wszystkich wystąpień ciągu `"adania"` ciągiem `"dania"`.

`"Zadania z programowania.".replaceAll("ania", "anka")` — zastąpienie wszystkich wystąpień ciągu `"ania"` ciągiem `"anka"`.

`"Zadania z programowania.".replaceFirst("ania", "anka")` — zastąpienie pierwszego wystąpienia ciągu `"ania"` ciągiem `"anka"`.

`"Zadania z programowania.".substring(10)` — wyodrębnienie ciągu od znaku o indeksie 10 do końca łańcucha wyjściowego.

`"Zadania z programowania.".substring(10, 17)` — wyodrębnienie ciągu od znaku o indeksie `10` do znaku stojącego przed znakiem o indeksie `17`. Długość wyodrębnionego łańcucha: *17–10 = 7* znaków.

Ebookpoint.pl kopia dla: pawher@o2.pl

`"Zadania z programowania.".concat("\b z podpowiedziami.")` — dołączenie na końcu łańcucha `"Zadania z programowania."` ciągu znaków `"\b z podpowiedziami."`. Pierwszy znak w dołączanym ciągu jest znakiem specjalnym `\b` (*backspace*) i spowoduje skasowanie ostatniego znaku (kropki) w łańcuchu wyjściowym.

`"Zadania z programowania."+"\b"+" z odpowiedziami."` — złączenie trzech łańcuchów znaków w jeden łańcuch (operator +).

Zadanie 4.2. DemoString.java, Z04_2.java

Tworzymy zmienną obiektową `napis` klasy `String` i inicjujemy ją wartością `"Zadania z programowania."`.

```
String napis = "Zadania z programowania.";
```

W tej sytuacji przykład z listingu 4.1 wygląda tak:

```
public class DemoString {
    public static void main(String args[]) {
        String napis = "Zadania z programowania.";
        System.out.println(napis);
        System.out.println(napis.charAt(0));
        System.out.println(napis.length());
        System.out.println(napis.charAt(napis.length()-1));
        System.out.println(napis.toUpperCase());
        System.out.println(napis.toLowerCase());
        System.out.println(napis.indexOf('z'));
        System.out.println(napis.indexOf("prog"));
        char kropka = '.', pytajnik = '?';
        System.out.println(napis.replace(kropka, pytajnik));
        System.out.println(napis.replace("adania", "dania"));
        String str1 = "ania", str2 = "anka";
        System.out.println(napis.replaceAll(str1, str2));
        System.out.println(napis.replaceFirst(str1, str2));
        System.out.println(napis.substring(10));
        System.out.println(napis.substring(10, 17));
        str1 = "\b z podpowiedziami.";
        System.out.println(napis.concat(str1));
        System.out.println(napis+str1.substring(0, 4)+str1.substring(5));
    }
}
```

Działanie zastosowanych metod omówiono w rozwiązaniu zadania 4.1.

Zadanie 4.3. WitajStr.java, Z04_3.java

Tworzymy obiekt `witaj` klasy `String` zainicjowany wartością `"Dzień dobry"`. Metoda `length` zwraca rozmiar (długość) łańcucha znaków. Znaki łańcucha są indeksowane (podobnie jak tablice) indeksami o wartościach od 0 do wartości o 1 mniejszej od długości łańcucha. Znaki łańcucha o podanym indeksie udostępnia nam metoda `charAt`.

W pętli `for(int i = 0; i < witaj.length(); ++i)...` uzyskamy dostęp (po kolei) do wszystkich znaków łańcucha `witaj.charAt(i)`. Znaki możemy w ten sposób wyłącznie odczytywać. Tak właśnie rozwiążemy części a) i b) naszego zadania.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class WitajStr {
    public static void main(String args[]) {
        String witaj = "Dzień dobry";
        /* Zadanie 4.3a */
        System.out.println("Pionowo: ");
        for(int i = 0; i < witaj.length(); ++i)
            System.out.println(witaj.charAt(i));
        System.out.println();
        /* Zadanie 4.3b */
        System.out.print("Tekst rozstrzelony: ");
        for(int i = 0; i < witaj.length(); ++i)
            System.out.print(witaj.charAt(i)+" ");
        System.out.println();
        /* Zadanie 4.3c */
        System.out.print("Wielkie litery: "+witaj.toUpperCase());
        System.out.println();
        /* Zadanie 4.3d */
        System.out.print("Małe litery: "+witaj.toLowerCase());
        System.out.println();
    }
}
```

W przypadku łańcucha znaków nie musimy zamieniać odrębnie każdego znaku (małe litery na wielkie lub odwrotnie, wielkie na małe). Do rozwiązania części zadania zawartych w podpunktach c) i d) możemy wykorzystać metody `toUpperCase` lub `toLowerCase` zwracające łańcuch (obiekt) zapisany odpowiednio wielkimi lub małymi literami.

Jeśli zamienimy łańcuch na tablicę znaków, to możemy skorzystać z pętli typu `for each` (podajemy fragment kodu) i rozwiązać zadanie w sposób znany z rozwiązania zadania 3.2:

```
/* Zadanie 4.3a */
System.out.println("Pionowo: ");
for(char z : witaj.toCharArray())
    System.out.println(z);
System.out.println();
/* Zadanie 4.3b */
System.out.print("Tekst rozstrzelony: ");
for(char z : witaj.toCharArray())
    System.out.print(z+" ");
System.out.println();
```

Zadanie 4.4. ProgramowanieStr.java, Z04_4.java, Z04_4a.java

Obiekt klasy `String` nie jest modyfikowalny, czyli nie jest możliwe zmienianie w nim znaków, dodawanie do niego nowych znaków lub usuwanie z niego znaków istniejących. Możemy natomiast zbudować na jego podstawie nowy łańcuch spełniający warunki zadania i przypisać go do tej samej referencji (zmiennej `nazwa`). Zbędny łańcuch zostanie usunięty z pamięci przez mechanizm *garbage collection* (ang. — czyszczenie pamięci, usuwanie niepotrzebnych obiektów).

Tworzymy łańcuch znaków `String napis = "programowanie";` zawierający słowo *programowanie* zapisane małymi literami.

Ebookpoint.pl kopia dla: pawher@o2.pl

a) Z łańcucha `napis` pobieramy pierwszy znak `napis.charAt(0)`, czyli znak *p*, i zamieniamy go na wielką literę `Character.toUpperCase(napis.charAt(0))`, otrzymując znak *P*. Do tego znaku dodajemy podłańcuch łańcucha `napis`, od znaku o indeksie *1* do końca łańcucha `napis.substring(1)`, czyli słowo *rogramowanie*. Operator *konkatenacji* (+) zamieni znak `'P'` na łańcuch jednoelementowy `"P"` i połączy z łańcuchem `"rogramowanie"` w łańcuch `"Programowanie"`:

```
napis = Character.toUpperCase(napis.charAt(0))+napis.substring(1);
```

Można to samo uzyskać nieco inaczej:

```
napis = (""+napis.charAt(0)).toUpperCase()+napis.substring(1);
```

W wyniku dodawania `""+napis.charAt(0)` (pusty łańcuch `""` łączymy z pojedynczym znakiem `'p'`) otrzymujemy jednoelementowy łańcuch `"p"`, który metoda `toUpperCase` zamienia na łańcuch `"P"`. Następnie mamy znaną już operację `"P"+"rogramowanie"`, oczywiście zapisaną w inny sposób.

I jeszcze raz to samo, ale w innym zapisie (komentarz należy do Czytelnika):

```
napis = napis.substring(0, 1).toUpperCase()+napis.substring(1);
```

b) Zamieniając wszystkie litery na wielkie, wystarczy użyć metody `toUpperCase` dla obiektu napis — `napis = napis.toUpperCase();`.

```
public class ProgramowanieStr {
    public static void main(String args[]) {
        String napis = "programowanie";
        System.out.print("Łańcuch znaków: ");
        System.out.println(napis);
        /* Zmiana pierwszej litery na wielką */
        napis = Character.toUpperCase(napis.charAt(0))+
            napis.substring(1);
        System.out.print("Pierwsza litera wielka: ");
        System.out.println(napis);
        /* Zmiana wszystkich liter na wielkie */
        napis = napis.toUpperCase();
        System.out.print("Wszystkie litery wielkie: ");
        System.out.println(napis);
    }
}
```

Powracając do podpunktu a) zadania, możemy zaproponować takie rozwiązanie:

```
String napis = "programowanie";
char[] znaki = napis.toCharArray();
znaki[0] = Character.toUpperCase(znaki[0]);
napis = String.valueOf(znaki);
```

Analizę kodu zostawiamy Czytelnikowi.

Zadanie 4.5. WspakStr1.java, Z04_5.java

Podobnie jak w rozwiązaniu zadania 3.4, wykorzystamy pętlę `for` ze zmniejszaniem wartości zmiennej sterującej.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class WspakStr1 {
    public static void main(String args[]) {
        String napis = "programowanie";
        System.out.print("Normalna kolejność znaków: ");
        System.out.println(napis);
        System.out.print("Odwrotna kolejność znaków: ");
        for(int i = napis.length()-1; i >= 0 ; --i)
             System.out.print(napis.charAt(i));
        System.out.println();
    }
}
```


Uwaga

Porównajmy fragment powyższego kodu z fragmentem rozwiązania zadania 3.4:

```
for(int i = dane.length-1; i >= 0; --i)
             System.out.print(dane[i]);
```

Obiekt `napis` klasy `String` ma metodę `length()` zwracającą liczbę znaków w łańcuchu, natomiast obiekt `dane` typu tablicowego (`char[]`) ma pole `length` zawierające liczbę elementów tablicy (w tym przypadku liczbę znaków). Ponadto inny jest sposób odczytywania znaków o podanym indeksie.

Zadanie 4.6. WspakStr2.java, Z04_6.java

Zadanie to możemy rozwiązać w trzech krokach: zamienić łańcuch znaków na tablicę znaków, odwrócić kolejność znaków w tablicy (zob. zadanie 3.5), zbudować łańcuch znaków na podstawie odwróconej tablicy.

```
public class WspakStr2 {
    public static void main(String args[]) {
        String napis = "programowanie";
        System.out.print("Normalna kolejność znaków: ");
        System.out.println(napis);
        /* Zamiana łańcucha na tablicę znaków */
        char[] dane = napis.toCharArray();
        /* Odwracanie tablicy znaków. */
        for(int i = 0, j = dane.length-1; i < j ; i++, j--) {
            char tmp = dane[i];
            dane[i] = dane[j];
            dane[j] = tmp;
        }
        /* Zamiana tablicy znaków na łańcuch */
        napis = new String(dane);
        System.out.print("Odwrotna kolejność znaków: ");
        System.out.println(napis);
    }
}
```

Do utworzenia łańcucha znaków na podstawie odwróconej tablicy wykorzystaliśmy konstruktor (`napis = new String(dane)`). Możemy zastosować metodę statyczną `valueOf` z klasy `String` (`napis = String.valueOf(dane)`).

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 4.7. CyfryStr.java, Z04_7.java

Odrzuciliśmy proste rozwiązanie `String cyfry = "0123456789";`. Możemy zbudować tablicę znaków zawierającą cyfry (zob. rozwiązanie zadania 3.7) i następnie zbudować na tej podstawie łańcuch znaków: `String cyfry = new String(tablica)`. Ponieważ zadanie ma bardziej charakter poznawczy niż praktyczny, proponujemy modyfikację rozwiązania zadania 3.7 w następujący sposób:

```
public class CyfryStr {
    public static void main(String args[]) {
        String cyfry = "";
        for(int i = 0; i < 10; ++i)
            cyfry += (char)(i+48);
        System.out.print("Cyfry układu dziesiątkowego: ");
        System.out.println(cyfry);
    }
}
```

Zwróćmy uwagę na operator += (podstawianie z dodawaniem — w tym przypadku dotyczące dodawania łańcuchów znaków), który skraca zapis wyrażenia. Zamiast `cyfry = cyfry+(char)(i+48)` piszemy krócej: `cyfry += (char)(i+48)`. Należy dodać, że w tym przykładzie do łańcucha znaków dodajemy jeden znak (konwersja znaku na łańcuch następuje automatycznie w czasie wykonywania tej operacji).

Zastosowane tutaj dodawanie łańcuchów jest przemienne, więc poprawny będzie zapis wyrażenia w postaci `cyfry = (char)(i+48)+cyfry` (nie mamy możliwości skrócenia wyrażenia za pomocą operatora +=). Jaki będzie efekt, Czytelnik może sprawdzić sam.


Uwaga

> Takiego zastosowania klasy `String` nie należy polecać. Każde dodawanie pojedynczego znaku (lub innego łańcucha) powoduje tworzenie nowego łańcucha z wynikiem (na szczęście nie musimy się troszczyć o ich usuwanie). Później poznasz klasy `StringBuffer` i `StringBuilder` lepiej nadające się do rozwiązywania podobnych zadań.

Zadanie 4.8. CyfryStr16.java, Z04_8.java

Do zbudowania łańcucha cyfr szesnastkowych wykorzystamy statyczną metodę `forDigit` z klasy `Character`.

```
public class CyfryStr16 {
    public static void main(String args[]) {
        String cyfry = "";
        for(int i = 0; i < 16; ++i)
            cyfry += Character.forDigit(i, 16);
        System.out.print("Cyfry układu szesnastkowego: ");
        cyfry = cyfry.toUpperCase();
        System.out.println(cyfry);
    }
}
```

Uwaga zamieszczona w rozwiązaniu zadania 4.7 pozostaje aktualna. Zamianę małych liter na wielkie możemy wykonać również podczas tworzenia łańcucha `cyfry` (w pętli):

```
cyfry += Character.toUpperCase(Character.forDigit(i, 16));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Możemy również skopiować kod rozwiązania zadania 3.8 i uzupełnić go o jedną linijkę kodu `String cyfryHex = String.valueOf(cyfry);` lub `String cyfryHex = new String(cyfry);`.

5. Tablica argumentów aplikacji

Zadanie 5.1. Argumenty.java, Z05_1.java

Po uruchomieniu aplikacji poleceniem `java Argumenty parametr1 parametr2 ... parametrN` system operacyjny przekazuje podane argumenty do aplikacji w postaci tablicy. Deklarację tej tablicy obowiązkowo umieszczamy w nagłówku metody `main`.

Do przeglądania tablicy argumentów możemy zastosować pętlę typu `for each`.

```
public class Argumenty {
    public static void main(String args[]) {
        System.out.println("Liczba argumentów "+args.length);
        /* Lista argumentów - pętla typu for each */
        for(String argument: args)
            System.out.println(argument);
    }
}
```

W Javie każda tablica jest obiektem, którego właściwość `length` zawiera liczbę elementów tablicy. Zatem pole `args.length` zawiera liczbę przekazanych argumentów. Argumenty w programie są dostępne przy użyciu indeksów od `0` do `args.length-1`: `args[0]`, `args[1]`,…, `args[N-1]`, gdzie `N` oznacza liczbę argumentów. Do przeglądania listy argumentów możemy zastosować pętlę typu `for`.

```
/* Lista argumentów - pętla typu for */
for(int i = 0; i < args.length; ++i)
    System.out.println(args[i]);
```

Zadanie 5.2. Osoba.java, Z05_2.java

Uruchamiamy w konsoli program poleceniem `java Osoba Maria Kowalska`. System operacyjny przekaże do tablicy argumentów dwa parametry (imię i nazwisko). W tablicy argumentów mamy zatem `args[0] = "Maria"` i `args[1] = "Kowalska"`. Zakładając (zgodnie z treścią zadania), że użytkownik może niedokładnie wprowadzić wielkość liter, musimy dokonać odpowiednich korekt danych przed ich wyświetlaniem.

`Nazwisko: Kowalska` — rozdzielamy `args[1]` na dwie części: pierwszy znak i reszta łańcucha; pierwszy znak zamienimy na wielką literę, a resztę łańcucha wyświetlimy małymi literami:

```
System.out.println(args[1].substring(0, 1).toUpperCase()
    +args[1].substring(1).toLowerCase());
```

`Imię: Maria` — rozdzielamy `args[0]` na dwie części: pierwszy znak i reszta łańcucha; pierwszy znak zamienimy na wielką literę, a resztę łańcucha wyświetlimy małymi literami:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
System.out.println(Character.toUpperCase(args[0].charAt(0))
    +args[0].substring(1).toLowerCase());
```

`Nazwisko i imię: KOWALSKA Maria` — nazwisko wyświetlamy wielkimi literami (`args[1].toUpperCase()`), imię tak jak w poprzednim punkcie.

`Inicjały: MK` — wyświetlamy pierwsze litery imienia i nazwiska:

```
System.out.print(Character.toUpperCase(args[0].charAt(0)));
System.out.println(Character.toUpperCase(args[1].charAt(0)));
```

`Login: KOmar` — login budujemy z dwóch początkowych liter nazwiska, zapisanych wielkimi literami (`args[1].substring(0, 2).toUpperCase()`) i trzech początkowych liter imienia zapisanych małymi literami (`args[0].substring(0, 3).toLowerCase()`).

```
import static java.lang.System.out;
public class Osoba {
    public static void main(String args[]) {
        out.print("Nazwisko: ");
        out.println(args[1].substring(0, 1).toUpperCase()+
            args[1].substring(1).toLowerCase());
        out.print("Imię: ");
        out.println(Character.toUpperCase(args[0].charAt(0))+
            args[0].substring(1).toLowerCase());
        out.print("Nazwisko i imię: "+args[1].toUpperCase()+" ");
        out.println(Character.toUpperCase(args[0].charAt(0))+
            args[0].substring(1).toLowerCase());
        out.print("Inicjały: ");
        out.print(Character.toUpperCase(args[0].charAt(0)));
        out.println(Character.toUpperCase(args[1].charAt(0)));
        out.print("Login: ");
        out.print(args[1].substring(0, 2).toUpperCase());
        out.println(args[0].substring(0, 3).toLowerCase());
    }
}
```

Zwróćmy uwagę na skrócenie wierszy programu poprzez import statycznego obiektu `out` z klasy `System` i pomijanie nazwy klasy w operacjach wyjścia.

W przykładzie pokazano również dwa sposoby zamienienia pierwszego znaku łańcucha na wielką literę: `str.toUpperCase().charAt(0)` — utworzono łańcuch zapisany wielkimi literami i pobrano pierwszy znak, `Character.toUpperCase(str.charAt(0))` — pobrano pierwszy znak łańcucha `str` i zamieniono na wielką literę.

W podanym rozwiązaniu nie zastosowano żadnych dodatkowych zmiennych łańcuchowych. Użycie pomocniczych zmiennych mogłoby zmniejszyć liczbę wykonywanych operacji na łańcuchach znaków. Pozostawimy to ćwiczenie do wykonania Czytelnikowi.

Zadanie 5.3. ArgsWspak.java, Z05_3.java

Wyświetlanie tablicy argumentów w odwrotnej kolejności możemy zrealizować w pętli `for` z odliczaniem w dół, od ostatniego do pierwszego elementu.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class ArgsWspak {
    public static void main(String args[]) {
        for(int i = args.length-1; i >= 0; --i)
            System.out.print(args[i]+" ");
        System.out.println("\b");
    }
}
```

Do każdego argumentu dodawany jest znak odstępu. Za ostatnim argumentem ten znak jest niepotrzebny (zresztą jest też niewidoczny, ale czasem mógłby przeszkadzać), więc kasujemy ten odstęp przy użyciu specjalnego znaku `'\b'` (ang. *backspace*).

Zadanie 5.4. ArgWspak.java, Z05_4.java

W pętli zewnętrznej przeglądamy zawartość tablicy argumentów, a w pętli wewnętrznej wyświetlamy argument w odwrotnej kolejności (wspak).

```
public class ArgWspak {
    public static void main(String args[]) {
        /* Przeglądanie tablicy argumentów - pętla zewnętrzna. */
        for(String argument : args) {
            /* Wyświetlanie argumentu wspak */
            for(int i = argument.length()-1; i >= 0; --i)
                System.out.print(argument.charAt(i));
            /* Przejście do nowego wiersza */
            System.out.println();
            }
    }
}
```

Zadanie 5.5. EtykietaArg.java, Z05_5.java

Podczas wywołania programu podajemy imię i nazwisko jako argumenty. Do trzeciego wiersza etykiety (`String wiersz = "*  *";`) musimy wstawić imię i nazwisko (`String tekst = args[0]+" "+args[1];`). Najpierw obliczamy różnicę długości tych napisów `int znaki = wiersz.length()-tekst.length();`. Z łańcucha `wiersz` bierzemy początek, połowę wyliczonej liczby znaków (`wiersz.substring(0, znaki/2)`). Do tego dodajemy imię i nazwisko (`tekst`) i na koniec dopisujemy resztę znaków, z końca łańcucha `wiersz` (`wiersz.substring(wiersz.length()-znaki+znaki/2)`).

```
public class EtykietaArg {
    public static void main(String args[]) {
        /* Wypiszemy pierwszy i drugi wiersz etykiety. */
        System.out.println("******* Programowanie *******");
        System.out.println("*  obiektowe w języku Java  *");

        /* Budujemy trzeci wiersz etykiety. Do tego wiersza musimy
           wstawić imię i nazwisko. */
        String wiersz =    "*                          *";
        String tekst = args[0]+" "+args[1];
        /* Obliczymy, ile znaków z oryginalnego wiersza należy
           pozostawić. */
        int znaki = wiersz.length()-tekst.length();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            /* Połowę pozostawionych znaków (początek wiersza) wstawimy
               przed tekstem. */
            String tmp = wiersz.substring(0, znaki/2)+tekst;
            /* Resztę znaków (koniec wiersza) dopiszemy po tekście. */
            tmp += wiersz.substring(wiersz.length()-znaki+znaki/2);
            /* Wypisujemy trzeci wiersz. */
            System.out.println(tmp);

            /* Pozostał do wypisania ostatni wiersz etykiety. */
            System.out.println("*****************************");
        }
    }
```

Występujące w obliczeniach dzielenie jest dzieleniem całkowitym, co oznacza, że nie zawsze wyrażenie `znaki-znaki/2` ma taką samą wartość jak wyrażenie `znaki/2`. Nieparzysta liczba znaków nie może być rozdzielona równomiernie przed tekstem i za tekstem. W takim przypadku na koniec zostawiamy o jeden znak więcej.

6. Prawda czy fałsz — logiczny typ danych

Zadanie 6.1. OperatoryLogiczne.java, Z06_1.java

Tworzymy tablicę `bool` zawierającą dwa elementy `false` i `true` typu `boolean`.

W pętli `for(boolean p: bool) {...}` zmienna logiczna `p` będzie przyjmowała kolejno wartości `false` i `true`. Dzięki temu będziemy mogli utworzyć tabelkę *negacji* i testować prawa logiczne z jednym zdaniem `p`.

Zagnieżdżając pętle `for(boolean p: bool) for(boolean q: bool) {...}`, otrzymamy parę zmiennych logicznych `p` i `q`, przyjmujących kolejno wartości: (`false`, `false`), (`false`, `true`), (`true`, `false`) i (`true`, `true`). Pozwoli nam to na utworzenie tabelek *koniunkcji*, *alternatywy* i innych funktorów logicznych dwuargumentowych oraz testowanie praw logicznych z dwoma zdaniami logicznymi *p* i *q*.

Powyższe uwagi zostaną wykorzystane w rozwiązaniach zadań 6.1, 6.2, 6.3, 6.4, 6.5 i 6.6 oraz innych podobnych.

Do sformatowania (w kolumnach) wyników operacji logicznych użyjemy znaku tabulatora `'\t'` wewnątrz łańcuchów tworzących nagłówki tabelek (aby uprościć kod, nie rysujemy linii, pozostawiając to Czytelnikowi) i łańcucha `"\t"` zawierającego znak tabulatora, łączonego z wartościami wyświetlanych zmiennych, np. `p+"\t"+!p`. W zadaniu wykorzystujemy operator *negacji* (`!`), *alternatywy* (`|`) i *koniunkcji* (`&`).

```
public class OperatoryLogiczne {
    public static void main(String args[]) {
        boolean[] bool = {false, true};

        System.out.println("Operator negacji (NOT) - !");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("  p\t  !p");
        for(boolean p: bool)
            System.out.println(p+"\t"+!p);
        System.out.println();

        System.out.println("Operator koniunkcji (AND) - & lub &&");
        System.out.println("  p\t  q\tp & q");
        for(boolean p: bool)
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+(p & q));
        System.out.println();

        System.out.println("Operator alternatywy (OR) - | lub ||");
        System.out.println("  p\t  q\tp | q");
        for(boolean p: bool)
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+(p | q));
        System.out.println();
    }
}
```


Uwaga

Operatory `&` i `|` są wykorzystywane również do przeprowadzania operacji bitowych na binarnej reprezentacji danych. Ponieważ `true` odpowiada liczbie `1`, a `false` liczbie `0` (w reprezentacji wewnętrznej), to obliczenia wykonane na wartościach logicznych dają właściwy wynik.

Operatory `&&` i `||` służą wyłącznie do działań na wartościach logicznych. Są nazywane szybkimi. Od operatorów `&` i `|` różnią się sposobem wyliczania wartości. Aby *alternatywa* dwóch wyrażeń była prawdziwa, wystarczy, że jeden z operandów alternatywy jest prawdziwy. Jeśli pierwszy operand alternatywy ma wartość `true`, to wiadomo, że alternatywa jest prawdziwa (wartości drugiego operandu nie musimy obliczać). W przypadku *koniunkcji* — jeśli pierwszy operand ma wartość `false`, to całe wyrażenie ma wartość `false`.

Zadanie 6.2. PrawaLogiczne.java, Z06_2.java

Badane prawa są zdaniami logicznymi zbudowanymi ze zdania `p` i poznanych w zadaniu 6.1 operatorów logicznych. Wartości zdania `p` pobieramy z tablicy `bool` i budujemy odpowiednie wyrażenia logiczne: `p || !p` — *prawo wyłączonego środka*, `!(p && !p)` — *prawo niesprzeczności*, `!(!p) == p)` — *prawo podwójnego przeczenia*. W ostatnim przypadku jako operator równoważności wykorzystano operator porównania `==`, który zwraca wartość `true` dla pary zdań równoważnych (`false == false`, `true == true`) i wartość `false` dla zdań nierównoważnych (`false == true`, `true == false`).

```
public class PrawaLogiczne {
    public static void main(String args[]) {
        boolean[] bool = {false, true};

        System.out.println("Prawo wyłączonego środka - p || !p");
        System.out.println("  p\t  !p\tp || !p");
        for(boolean p: bool)
            System.out.println(p+"\t"+!p+"\t"+(p || !p));
        System.out.println();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Prawo niesprzeczności - !(p && !p)");
        System.out.println("  p\t  !p\tp && !p\t!(p && !p)");
        for(boolean p: bool)
            System.out.println(p+"\t"+!p+"\t"+(p && !p)+"\t"+!(p && !p));
        System.out.println();

        System.out.println("Prawo podwójnego przeczenia - !(!p) == p)");
        System.out.println("  p\t  !p\t!(!p)\t!(!p) == p)");
        for(boolean p: bool)
            System.out.println(p+"\t"+!p+"\t"+!(!p)+"\t"+(!(!p) == p));
        System.out.println();
    }
}
```

Zadanie 6.3. PrawaDeMorgana.java, Z06_3.java

Stosując tablicę `bool` i zagnieżdżone pętle (zob. komentarz w rozwiązaniu zadania 6.1), możemy zbadać *prawa De Morgana*. W dwóch początkowych kolumnach umieszczamy wszystkie możliwe wartości dla dwóch zdań logicznych (`p` i `q`). W kolejnych umieszczamy wartości pośrednich wyników, a w ostatniej kolumnie wartość całego zdania (kompletne wyrażenie). Uzyskane w ostatniej kolumnie wartości `true` świadczą, że to wyrażenie jest prawem logicznym.

```
public class PrawaDeMorgana {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Zaprzeczenie koniunkcji jest równoważne
            alternatywie zaprzeczeń");
        System.out.println("p\tq\tp&&q\t!(p&&q)\t!p\t!q\t!p||!q\t!(p&&q)
            <=>(!p||!q)");
        for(boolean p: bool)
            for(boolean q: bool) {
                System.out.print(p+"\t"+q+"\t"+(p&&q)+"\t"+!(p&&q));
                System.out.print("\t"+!p+"\t"+!q+"\t"+(!p||!q));
                System.out.println("\t"+(!(p&&q)==(!p||!q)));
                }
        System.out.println("\n");
        System.out.println("Zaprzeczenie alternatywy jest równoważne
            koniunkcji zaprzeczeń");
        System.out.println("p\tq\tp||q\t!(p||q)\t!p\t!q\t!p&&!q\t!(p||q)
            <=>(!p&&!q)");
        for(boolean p: bool)
            for(boolean q: bool) {
                System.out.print(p+"\t"+q+"\t"+(p||q)+"\t"+!(p||q));
                System.out.print("\t"+!p+"\t"+!q+"\t"+(!p&&!q));
                System.out.println("\t"+(!(p||q)==(!p&&!q)));
                }
        System.out.println("\n");
    }
}
```

Zadanie 6.4. Implikacja.java, Z06_4.java, Z06_4a.java

Wykorzystując rozwiązanie zadania 6.2, uwagi z rozwiązania zadania 6.1 oraz podaną wskazówkę, utworzymy następującą aplikację:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Implikacja {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Implikacja - jeżeli p, to q");
        System.out.println("p\tq\tp=>q");
        for(boolean p: bool)
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+
                    (Boolean.valueOf(p).compareTo(q) < 1));
        System.out.println();
    }
}
```


Uwaga

Możemy zdefiniować własną metodę statyczną realizującą zadania operatora implikacji (proponujemy nazwę `impl`). Więcej szczegółów o definiowaniu metod poznasz dalej.

```
static boolean impl(boolean p, boolean q) {
    return (Boolean.valueOf(p).compareTo(q) < 1));
}
```

Po takiej zmianie otrzymamy następujący kod:

```
public class Implikacja {
    static boolean impl(boolean p, boolean q) {
        return (Boolean.valueOf(p).compareTo(q) < 1);
    }
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Implikacja - jeżeli p, to q");
        System.out.println("p\tq\tp=>q");
        for(boolean p: bool)
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+ impl(p, q));
        System.out.println();
    }
}
```

Zadanie 6.5. Xor.java, Z06_5.java

Alternatywa wykluczająca (*p albo q*) realizowana jest przez operator *XOR* (^) — alternatywa wykluczająca (ang. *exclusive or*). Możemy również skorzystać z operatora *nie jest równe* (!=), ponieważ alternatywa wykluczająca jest prawdziwa, gdy jedno ze zdań jest prawdziwe, a drugie fałszywe, czyli gdy ich wartości logiczne nie są równe.

W rozwiązaniu zadania pokażemy i porównamy działanie operatora *XOR* (^) oraz operatora *nie jest równe* (!=) dla wartości logicznych.

```
public class Xor {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Operator alternatywy wykluczającej (XOR)
            - ^");
        System.out.println("  p\t  q\tp ^ q");
        for(boolean p: bool)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+(p ^ q));
        System.out.println();
        System.out.println("Operator nie jest równe - !=");
        System.out.println("  p\t  q\tp != q");
        for(boolean p: bool)
            for(boolean q: bool)
                System.out.println(p+"\t"+q+"\t"+(p != q));
        System.out.println();
    }
}
```

Zadanie 6.6. Z06_6a.java, Z06_6b.java, Z06_6b.java, Z06_6d.java, Z06_6e.java

W prawach dowodzonych w podpunktach a), b) i c) mamy prawa logiczne z trzema zdaniami logicznymi `p`, `q` i `r`. Użyjemy trzech zagnieżdżonych pętli `for` i tablicy `bool` do wygenerowania wszystkich możliwych trójek wartości logicznych dla zdań logicznych `p`, `q` i `r`.

Prawa dowodzone w podpunktach d) i e) zawierają tylko dwa zdania logiczne `p` i `q`. W zadaniach a), d) i e) występuje *implikacja*. Skorzystamy z metody `impl` wprowadzonej w rozwiązaniu zadania 6.4.

W tabelach wprowadzono dodatkowe oznaczenia A, B, C i D, skracające opis nagłówka tabeli.

a) Prawo przechodniości implikacji $[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$.

```
public class PrawoLogiczneA {
    static boolean impl(boolean p, boolean q) {
        return (Boolean.valueOf(p).compareTo(q) < 1);
    }
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Prawo przechodniości implikacji");
        System.out.println("p\tq\tr\tp=>q\tq=>r\tA & B\tp=>r\tC=>D");
        System.out.println("\t\t\t(A)\t(B)\t(C)\t(D)");
        for(boolean p: bool) for(boolean q: bool) for(boolean r: bool) {
            System.out.print(p+"\t"+q+"\t"+r);
            System.out.print("\t"+impl(p, q)+"\t"+impl(q,r));
            System.out.print("\t"+(impl(p, q)&impl(q, r)));
            System.out.print("\t"+impl(p,r)+"\t");
            System.out.println(impl(impl(p, q)&impl(q, r), impl(p,r)));
        }
        System.out.println();
    }
}
```

b) Prawo rozdzielności alternatywy względem koniunkcji $[p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$. W celu uproszczenia zapisu wprowadzono dodatkowe zmienne `a`, `b`, `c`, `d` i `e` typu `boolean`.

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

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        boolean[] bool = {false, true};
        System.out.println("Prawo rozdzielności alternatywy względem
            koniunkcji\n");
        System.out.println("p\tq\tr\tq&r\tp|A\tp|q\tp|r\tC&D\tB<=>E ");
        System.out.println("\t\t\t(A)\t(B)\t(C)\t(D)\t(E)");
        for(boolean p: bool) for(boolean q: bool) for(boolean r: bool) {
            System.out.print(p+"\t"+q+"\t"+r);
            boolean a = q&r;
            boolean b = p|a;
            boolean c = p|q;
            boolean d = p|r;
            boolean e = c&d;
            System.out.print("\t"+a+"\t"+b+"\t"+c+"\t"+d+"\t"+e+"\t");
            System.out.println(b == e);
        }
        System.out.println();
    }
}
```

c) Prawo rozdzielności koniunkcji względem alternatywy $[p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$. Podobnie jak w podpunkcie b), w celu uproszczenia zapisu wprowadzono dodatkowe zmienne `a`, `b`, `c`, `d` i `e` typu `boolean`.

```
public class PrawoLogiczneC {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Prawo rozdzielności koniunkcji względem
            alternatywy\n");
        System.out.println("p\tq\tr\tq|r\tp&A\tp&q\tp&r\tC|D\tB<=>E");
        System.out.println("\t\t\t(A)\t(B)\t(C)\t(D)\t(E)");
        for(boolean p: bool) for(boolean q: bool) for(boolean r: bool) {
            System.out.print(p+"\t"+q+"\t"+r);
            boolean a = q|r;
            boolean b = p&a;
            boolean c = p&q;
            boolean d = p&r;
            boolean e = c|d;
            System.out.print("\t"+a+"\t"+b+"\t"+c+"\t"+d+"\t"+e+"\t");
            System.out.println(b == e);
        }
        System.out.println();
    }
}
```

d) Prawo odrywania $[(p \Rightarrow q) \wedge q] \Rightarrow q$.

Zbudujemy dodatkową klasę `Impl` zawierającą definicję metody `impl` zbudowanej w rozwiązaniu zadania 6.4. W pliku *Impl.java* wpisujemy kod:

```
public class Impl {
    static boolean impl(boolean p, boolean q) {
        return (Boolean.valueOf(p).compareTo(q) < 1);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Po skompilowaniu otrzymamy plik *Impl.class*. Jeśli ten plik umieścimy w bieżącym folderze, to metoda statyczna `impl` będzie dostępna dla wszystkich klas znajdujących się w tym samym folderze, czyli w tzw. *pakiecie domyślnym*. Jest to w tym zbiorze zadań pierwszy przykład klasy niezawierającej metody `main()`, czyli niebędącej aplikacją. Metodę będziemy wywoływać z innych klas, np. `Impl.impl(p, q)`.

```
public class PrawoLogicznеD {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.println("Prawo odrywania [(p => q) & q] => q\n");
        System.out.println("p\tq\tp=>q\tA & q\tB=>q");
        System.out.println("\t\t(A)\t(B)");
        for(boolean p: bool) for(boolean q: bool) {
            boolean a = Impl.impl(p, q);
            boolean b = a & q;
            System.out.print(p+"\t"+q+"\t"+a+"\t"+b+"\t");
            System.out.println(Impl.impl(b, q));
        }
        System.out.println();
    }
}
```

e) Prawo eliminacji implikacji $(p \Rightarrow q) \Leftrightarrow (\neg p \vee q)$. Podobnie jak w punkcie d), skorzystamy z metody `impl` (zastępującej operator implementacji) zdefiniowanej w klasie `Impl`.

```
public class PrawoLogicznеE {
    public static void main(String args[]) {
        boolean[] bool = {false, true};
        System.out.print("Prawo eliminacji implikacji ");
        System.out.println("(p => q) <=> (!p|q)\n");
        System.out.println("p\tq\tp=>q\t!p\t!p|q\tA<=>B");
        System.out.println("\t\t(A)\t\t(B)");
        for(boolean p: bool) for(boolean q: bool) {
            boolean a = Impl.impl(p, q);
            boolean b = !p | q;
            System.out.print(p+"\t"+q+"\t"+a+"\t"+(!p)+"\t"+b+"\t");
            System.out.println(a == b);
        }
        System.out.println();
    }
}
```

Zadanie 6.7. StringIsEmpty.java, Z06_7.java

Metoda `isEmpty` zwraca wartość `true`, gdy łańcuch nie zawiera ani jednego znaku (jest pusty), i wartość `false`, gdy łańcuch zawiera co najmniej jeden znak.

```
public class StringIsEmpty {
    public static void main(String[] args) {
        String str1 = "";
        System.out.println("String: "+str1);
        System.out.println("Długość łańcucha: "+str1.length());
        System.out.println("Czy string jest pusty? "+str1.isEmpty());
        str1 = "Krótki tekst...";
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("String: "+str1);
        System.out.println("Długość łańcucha: "+str1.length());
        System.out.println("Czy string jest pusty? "+str1.isEmpty());
    }
}
```

Zadanie 6.8. TestChar.java, Z06_8.java

a) Ciąg składa się z 7 znaków, są to: wielka litera `A`, tzw. spacja nierozdzielająca (kod dziesiętny *160*, ósemkowy `\240`), mała litera `b`, cyfra `3`, znak `&`, cyfra `4` i odstęp (spacja — kod dziesiętny 32, biały znak).

Program testujący wyświetla znaki występujące w łańcuchu, a następnie tworzy tabelę z wynikami testów z użyciem metod `isDigit()`, `isLetter()`, `isLetterOrDigit()`, `isLowerCase()`, `isSpaceChar()`, `isUpperCase()` i `isWhiteSpace()`. W każdej kolumnie znajdują się znak, kod znaku i wyniki testu dla tego znaku.

```
public class TestChar {
    public static void main(String[] args) {
        String str = "A\240b3&4\040"; // podpunkt a)
        System.out.println("Znaki: "+str);
        char[] znaki = str.toCharArray();
        System.out.print("Znak             ");
        for(char z : znaki)
            System.out.print("\t"+z);
        System.out.println();
        System.out.print("Kod znaku        ");
        for(char z : znaki)
            System.out.print("\t"+(int)z);
        System.out.println();
        System.out.print("isDigit()        ");
        for(char z : znaki)
            System.out.print("\t"+Character.isDigit(z));
        System.out.println();
        System.out.print("isLetter()       ");
        for(char z : znaki)
            System.out.print("\t"+Character.isLetter(z));
        System.out.println();
        System.out.print("isLetterOrDigit()");
        for(char z : znaki)
            System.out.print("\t"+Character.isLetterOrDigit(z));
        System.out.println();
        System.out.print("isLowerCase()    ");
        for(char z : znaki)
            System.out.print("\t"+Character.isLowerCase(z));
        System.out.println();
        System.out.print("isSpaceChar()    ");
        for(char z : znaki)
            System.out.print("\t"+Character.isSpaceChar(z));
        System.out.println();
        System.out.print("isUpperCase()    ");
        for(char z : znaki)
            System.out.print("\t"+Character.isUpperCase(z));
        System.out.println();
        System.out.print("isWhitespace()   ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(char z : znaki)
                System.out.print("\t"+Character.isWhitespace(z));
            System.out.println();
        }
    }
```

Po uruchomieniu programu uzyskamy wynik:

```
Znaki: A b3&4
Znak                A       b       3       &       4
Kod znaku           65      160     98      51      38      52      32
isDigit()           false   false   false   true    false   true    false
isLetter()          true    false   true    false   false   false   false
isLetterOrDigit()   true    false   true    true    false   true    false
isLowerCase()       false   false   true    false   false   false   false
isSpaceChar()       false   true    false   false   false   false   true
isUpperCase()       true    false   false   false   false   false   false
isWhitespace()      false   false   false   false   false   false   true
```

b) Ciąg składa się z 7 znaków, są to: wielka litera Ł, mała litera o, mała litera ś, znak podkreślenia _, cyfra 0, znak + i znak tabulatora (\t — kod dziesiętny 9). W kodzie programu *TestChar* zmieniamy jeden wiersz: `String str = "Łoś_4-\t";`. Dla tego łańcucha otrzymamy:

```
Znaki: Łoś_0+
Znak                Ł       o       ś       _       0       +
Kod znaku           321     111     347     95      48      43
isDigit()           false   false   false   false   true    false
isLetter()          true    true    true    false   false   false
isLetterOrDigit()   true    true    true    false   true    false
isLowerCase()       false   true    true    false   false   false
isSpaceChar()       false   false   false   false   false   false
isUpperCase()       true    false   false   false   false   false
isWhitespace()      false   false   false   false   false   false
```

Prezentowane znaki są znakami Unicode, co wyraźnie widać w przypadku polskich znaków *Ł* i *ś* (kody większe od 255, dwubajtowe).

c) Ciąg składa się z 7 znaków, są to: znak #, znak cudzysłowu " (znak specjalny \"), wielka litera Ä (kod ósemkowy \304), znak \ (znak specjalny \\), mała litera ä (kod ósemkowy \344), znak *backspace* (znak specjalny \b, kod dziesiętny 8) i znak *newline* (znak specjalny \n, kod dziesiętny 10).

W kodzie programu podstawimy wiersz `String str = "#\"\304\\\344\b\n";` i uzyskamy wynik:

```
Znaki: #"Ä\ä

Znak                #       "       Ä       \       ä

Kod znaku           35      34      196     92      228     8       10
isDigit()           false   false   false   false   false   false   false
isLetter()          false   false   true    false   true    false   false
isLetterOrDigit()   false   false   true    false   true    false   false
isLowerCase()       false   false   false   false   true    false   false
isSpaceChar()       false   false   false   false   false   false   false
isUpperCase()       false   false   true    false   false   false   false
isWhitespace()      false   false   false   false   false   false   true
```

Ebookpoint.pl kopia dla: pawher@o2.pl

7. Liczby całkowite typu int i klasa Integer

Zadanie 7.1. StaticInteger.java, Z07_1.java

Objaśnienia stałych i wybranych metod statycznych klasy `Integer` opakowującej typ prosty `int` zawarto w tekście programu.

```
public class StaticInteger {
    public static void main(String args[]) {
        System.out.println("Wybrane stałe i metody statyczne klasy
            Integer\n");
        System.out.println("Wartość minimalna typu int: "+
            Integer.MIN_VALUE);
        System.out.println("Wartość maksymalna typu int: "+
            Integer.MAX_VALUE);
        System.out.println("Rozmiar (w bitach) typu int: "+Integer.SIZE);
        int a = 179;
        System.out.println("Liczba całkowita a = "+a+
            " - zapis (ciąg cyfr):");
        System.out.println("w systemie binarnym: "+
            Integer.toBinaryString(a));
        System.out.println("w systemie ósemkowym: "+
            Integer.toOctalString(a));
        System.out.println("w systemie szesnastkowym: "+
            Integer.toHexString(a));
        System.out.println("w systemie dziesiątkowym: "+
            Integer.toString(a));
        System.out.println("w systemie czwórkowym: "+
            Integer.toString(a, 4));
        int b = Integer.parseInt("-177");
        System.out.println("b = "+b);
        int c = Integer.parseInt("-177", 8);
        System.out.println("c = "+c);
        System.out.println("Metoda signum():");
        System.out.println("Znak liczby a: "+Integer.signum(a));
        System.out.println("Znak liczby b: "+Integer.signum(b));
        System.out.println("Znak liczby 0: "+Integer.signum(0));
    }
}
```

Zwróćmy uwagę na możliwość parsowania tego samego ciągu znaków (np. *"-177"*) w układach pozycyjnych o różnej podstawie. Skompilujmy i uruchommy ten program dla różnych wartości zmiennej `a`. Porównajmy sposób zapisu liczby dodatniej i ujemnej (np. *179* i *–179*) w różnych systemach liczbowych.

Zadanie 7.2. ObjectInteger.java, Z07_2.java

Objaśnienia metod zawarto w kodzie lub komentarzach do kodu.

```
public class ObjectInteger {
    public static void main(String args[]) {
        System.out.println("Wybrane metody obiektów klasy Integer\n");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        /* Konstruktor tworzący nowy obiekt typu Integer reprezentujący
           podaną liczbę, w tym przypadku 1024. */
        Integer a = new Integer(1024);
        /* Konstruktor tworzący nowy obiekt klasy Integer na podstawie
           podanego łańcucha znaków (ciągu cyfr). Zero nieznaczące
           zostanie pominięte. */
        Integer b = new Integer("02000");
        /* Statyczna metoda decode() zwraca obiekt reprezentujący liczbę
           zakodowaną w łańcuchu znaków - 0 na początku oznacza liczbę
           zapisaną w systemie ósemkowym. */
        Integer c = Integer.decode("02000");
        /* Jw. - początek ciągu w postaci 0x oznacza liczbę zakodowaną
           w systemie szesnastkowym. */
        Integer d = Integer.decode("0x2000");
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("c = "+c);
        System.out.println("d = "+d);
        /* Metoda equals() sprawdza, czy obiekt wywołujący tę metodę jest
           równy obiektowi podanemu jako parametr. Wynik porównania jest
           wartością logiczną (true albo false). */
        System.out.println("Czy obiekt a jest równy obiektowi b? "+
            a.equals(b));
        System.out.println("Czy obiekt a jest równy obiektowi c? "+
            a.equals(c));
        /* Metoda compareTo() porównuje obiekt  wywołujący tę metodę
           z obiektem stanowiącym parametr. Wynik jest jedną z liczb: -1,
           0 i 1. 0 oznacza równość obiektów, -1 oznacza, że pierwszy obiekt (wywołujący
           metodę) jest mniejszy od drugiego obiektu (parametru),
           a 1 oznacza, że pierwszy obiekt jest większy od drugiego). */
        System.out.println("Porównanie obiektu a z obiektem c? "+
           a.compareTo(c));
        System.out.println("Porównanie obiektu c z obiektem d? "+
           c.compareTo(d));
        System.out.println("Porównanie obiektu d z obiektem c? "+
           d.compareTo(c));
        /* Metoda statyczna valueOf() zwraca obiekt klasy Integer
           reprezentujący wartość określoną jako parametr - liczba
           całkowita lub łańcuch znaków (cyfr dziesiętnych). */
        a = Integer.valueOf(1000);
        b = Integer.valueOf("1000");
        /* Drugi parametr metody statycznej valueOf określa podstawę
           systemu liczbowego, w którym należy zinterpretować ciąg cyfr
           podany jako pierwszy parametr metody. */
        c = Integer.valueOf("1000", 2);  // 1000 w układzie binarnym
        d = Integer.valueOf("1000", 16); // 1000 w układzie szesnastkowym
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("c = "+c);
        System.out.println("d = "+d);
        /* Tu wstaw ponownie kod do porównywania obiektów. */
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Metoda `obj1.equals(obj2)` zwraca wartość `false`, gdy obiekty mają różne wartości, i `true` w przypadku równych wartości wszystkich pól obiektów `obj1` i `obj2` (o ile w implementacji metody `equals()` nie zdecydujemy inaczej). Użycie operatora porównania `obj1 == obj2` zwróci nam wynik porównania referencji obiektów, a nie ich wartości.

Metoda `obj1.compareTo(obj2)` zwraca `0`, gdy obiekty są równe; `-1`, gdy wartość `obj1` jest mniejsza od wartości `obj2`, albo `1`, gdy wartość `obj1` jest większa od wartości `obj2`.

Zadanie 7.3. Dodawanie.java, Z03_3.java

Na listingu przedstawiono trzy warianty rozwiązania zadania (w jednym pliku), różniące się sposobem obliczenia i wyświetlenia wyniku.

W pierwszym przypadku najpierw jest wyświetlany tekst `"a + b = "` przy użyciu metody `print()`, a potem obliczana jest wartość wyrażenia `a+b`. Wynik przekazany jest jako parametr wywołania metody `println()`.

W drugim przypadku następuje połączenie (dodawanie) tekstu z wynikiem dodawania wartości zmiennych. Ponieważ działania dodawania wykonywane są od strony lewej do prawej, to do tekstu dodana jest wartość zmiennej `a` (skonwertowanej na tekst), a następnie do wyniku (będącego łańcuchem znaków) dodawana jest druga liczba `b` (zamieniona na tekst). W efekcie zamiast sumy dwóch liczb otrzymujemy sumę łańcuchów cyfr reprezentujących te liczby. Aby uzyskać prawidłowy rezultat, należy zmienić kolejność wykonywania działań (użyć nawiasów). Najpierw będzie obliczona suma wartości zmiennych `(a+b)`, a następnie wynik zostanie zamieniony na łańcuch znaków i dołączony z prawej strony tekstu `"a + b ="`.

W trzecim przypadku używamy dodatkowej zmiennej `c` do przechowania sumy `a+b`. Wyświetlenie wyniku nie sprawia nam żadnych niespodzianek.

```
public class Dodawanie {
    public static void main(String args[]) {
        int a = 2451, b = 375;
        System.out.println("a = "+a+", b = "+b);
        /* wariant 1. */
        System.out.print("a + b = ");
        System.out.println(a+b);
        /* wariant 2. */
        System.out.println("a + b = "+a+b); // błąd
        System.out.println("a + b = "+(a+b));
        /* wariant 3. */
        int c = a+b;
        System.out.println("a + b = "+c);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 7.4. Z07_4a.java, Z07_4b.java, Z07_4c.java, Z07_4d.java

a) *Odejmowanie.java* — wzorując się na rozwiązaniu zadania 7.3, zbudujemy program:

```
public class Odejmowanie {
    public static void main(String args[]) {
        int a = 2451, b = 375;
        System.out.println("a = "+a+", b = "+b);
        /* Wariant 1. - osobne wyświetlanie tekstu i liczby. */
        System.out.print("a - b = ");
        System.out.println(a-b);
        /* Wariant 2. - błędny wynik. */
        // System.out.println("a - b = "+a-b); // źle
        System.out.println("a - b = "+(a-b));  // dobrze
        /* Wariant 3. - wprowadzamy dodatkową zmienną c. */
        int c = a-b;
        System.out.println("a - b = "+c);
    }
}
```

Zauważmy, że w rozwiązaniu zadania 7.3 instrukcja `System.out.println("a + b = "+a+b);` dawała błędny wynik (w stosunku do treści zadania), natomiast w przypadku odejmowania instrukcja o postaci `System.out.println("a – b = "+a-b);` się nie skompiluje. Nie ma odejmowania łańcuchów znaków! Ponieważ operatory + i - mają ten sam priorytet, należy użyć nawiasów w celu zmiany kolejności wykonywania działań `System.out.println("a – b = "+(a-b));`.

b) *Mnożenie.java* — program ten budujemy w podobny sposób jak program 7.3 i 7.4a.

```
public class Mnożenie {
    public static void main(String args[]) {
        int a = 2451, b = 375;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a * b = "+a*b);
    }
}
```

Na powyższym listingu przedstawiono tylko jeden z możliwych wariantów rozwiązania zadania (zob. rozwiązania zadań 7.3 i 7.4a). Uwaga przedstawiona w rozwiązaniu zadania 7.4a nie jest już aktualna — operator mnożenia * ma wyższy priorytet niż operator +, więc użycie nawiasu nie jest konieczne. Kod w tej postaci skompiluje się i obliczenia zostaną wykonane poprawnie.

c) *Dzielenie.java* — kod jest zupełnie podobny do tego z rozwiązania zadania 7.4b (operator * zastąpiono operatorem /).

```
public class Dzielenie {
    public static void main(String args[]) {
        int a = 2451, b = 375;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a / b = "+a/b);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Operator dzielenia ma wyższy priorytet od operatora dodawania, więc użycie nawiasów nie jest konieczne (zob. uwagi do rozwiązań zadań 7.3, 7.4a i 7.4b).

d) *Reszta.java* — wyrażenie `a%b` oblicza resztę z dzielenia liczby całkowitej `a` przez liczbę całkowitą `b`. Operator `%` ma wyższy priorytet od operatora (+).

```
public class Reszta {
    public static void main(String args[]) {
        int a = 2451, b = 375;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a % b = "+a%b);
        System.out.println("b % a = "+b%a);
    }
}
```

Zadanie 7.5. Dzielenie2.java, Z07_5.java

Dzielenie z resztą wykonujemy dla wybranej pary liczb. Zmieniamy znaki tych liczb tak, aby uzyskać wszystkie możliwe pary znaków dzielnej i dzielnika.

```
public class Dzielenie2 {
    public static void main(String args[]) {
        int a = 153, b = 15;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a : b = "+a/b+" r. "+a%b+"\n");
        b = -15;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a : b = "+a/b+" r. "+a%b+"\n");
        a = -153;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a : b = "+a/b+" r. "+a%b+"\n");
        b = 15;
        System.out.println("a = "+a+", b = "+b);
        System.out.println("a : b = "+a/b+" r. "+a%b+"\n");
    }
}
```

Zauważmy, że reszta z dzielenia przyjmuje znak dzielnej, natomiast znak ilorazu zależy od znaków dzielnej i dzielnika (iloraz jest dodatni, gdy dzielna i dzielnik mają ten sam znak — ujemny, gdy operandy mają różne znaki).

Zadanie 7.6. Suma.java, Z07_6.java

Z tablicy argumentów odczytujemy dwa argumenty: `args[0]` i `args[1]`. Zamieniamy łańcuchy znaków na liczby całkowite (stosujemy statyczną metodę `parseInt` z klasy `Integer`) i obliczamy ich sumę.

```
public class Suma {
    public static void main(String args[]) {
        int suma = Integer.parseInt(args[0])+Integer.parseInt(args[1]);
        System.out.println("Suma S = "+suma);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Aplikację uruchamiamy z dwoma parametrami, np. `java Suma 231 -347`. Jeśli podczas uruchomienia nie podamy dwóch argumentów lub argumenty nie będą poprawnie zapisanymi liczbami, to aplikacja przerwie pracę i pojawi się odpowiedni komunikat.

Zadanie 7.7. Sumuj.java

Aplikację uruchamiamy z kilkoma parametrami, np. `java Sumuj 127 -242 391`. Deklarujemy i inicjujemy zmienną całkowitą `int suma = 0;`. Odczytujemy kolejne argumenty z tablicy `args` w pętli `for`, zamieniamy je na liczby całkowite i dodajemy do zmiennej `suma`.

```
public class Sumuj {
    public static void main(String args[]) {
        int suma = 0;
        for(String argument: args)
            suma += Integer.parseInt(argument);
        System.out.println("Suma S = "+suma);
    }
}
```

W obliczeniach możemy również posłużyć się metodami z klasy `Integer`:

```
suma += Integer.valueOf(argument).intValue();
```

Najpierw tworzymy obiekt `Integer.valueOf(argument)`, a następnie przy wykorzystaniu metody `intValue()` zwracamy wartość jedynego pola obiektu (typ `int`).

Taki sam efekt uzyskamy z użyciem metody `decode()`:

```
suma += Integer.decode(argument).intValue();
```

W tym przypadku argumenty można podawać również w postaci szesnastkowej (np. `0xAF`, `0XAF` lub `#1AF0`) albo ósemkowej (`0742`).

Możemy w wyrażeniu sumującym użyć polecenia `new` i konstruktora:

```
suma += new Integer(argument).intValue();
```

Gdy pominiemy metodę `intValue()` w powyższych wariantach rozwiązania, nie spowoduje to błędu (w Javie od wersji 5. wprowadzono autoboxing). Podczas kompilacji wyrażenia obiekt klasy `Integer` jest automatycznie zamieniany na wartość typu `int`.

Jeśli uruchomimy aplikację bez argumentów, to otrzymamy sumę *0*. Jeśli natomiast któryś z parametrów nie będzie liczbą całkowitą, to działanie programu zostanie przerwane.

Zadanie 7.8. Zamiana.java, Z07_8.java

Aplikację uruchamiamy z dwoma parametrami — pierwszy parametr interpretujemy jako podstawę systemu (`int radix = Integer.parseInt(args[0])`), a drugi będzie zamienianą liczbą (`int n = Integer.parseInt(args[1])`). Stosując metodę `toString` z klasy `Integer`, zamieniamy liczbę całkowitą `n` na łańcuch znaków `str` reprezentujący tę liczbę w systemie pozycyjnym o podstawie `radix`. Podstawa `radix` powinna być liczbą z zakresu od `2` do `36`; wyjście poza ten zakres nie spowoduje błędów — zostanie przyjęta domyślna wartość podstawy (`radix = 10`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Zamiana {
    public static void main(String args[]) {
        int radix = Integer.parseInt(args[0]);
        int n = Integer.parseInt(args[1]);
        String str = Integer.toString(n, radix);
        System.out.println(args[1]+"[10] = "+str+"["+radix+"]");
    }
}
```

Uruchamiając aplikację, np. `java Zamiana 2 255`, otrzymamy wynik `255[10] = 1111 1111[2]`.

Zadanie 7.9. Zamiana2.java, Z07_9.java

Pierwszy parametr interpretujemy jako podstawę systemu (`int radix = Integer.parse Int(args[0])`), a drugi będzie ciągiem cyfr w systemie pozycyjnym o podstawie `radix`. Stosując metodę `parseInt` z klasy `Integer`, zamieniamy łańcuch znaków `args[1]` reprezentujący liczbę w systemie o podstawie `radix` na liczbę w systemie dziesiątkowym (`int n = Integer.parseInt(args[1], radix)`).

```
public class Zamiana2 {
    public static void main(String args[]) {
        int radix = Integer.parseInt(args[0]);
        int n = Integer.parseInt(args[1], radix);
        System.out.println(args[1]+"["+radix+"] = "+n+"[10]");
    }
}
```

Wywołując aplikację, np. `java Zamiana2 2 11111111`, otrzymamy wynik `11111111[2] = 255[10]`. Podanie podstawy wykraczającej poza dopuszczalny zakres lub ciągów znaków niebędących liczbami w wybranym systemie spowoduje przerwanie programu.

8. Inne typy liczb całkowitych w Javie

Zadanie 8.1. LongNumber.java, Z08_1.java, Z08_1a.java

Po skompilowaniu i uruchomieniu aplikacji (z listingu 8.1) otrzymamy w konsoli taki wynik:

```
n = 9223372036854775807
BIN: 1111111111111111111111111111111111111111111111111111111111111111
HEX: 7fffffffffffffff
m = -9223372036854775808
BIN: 1000000000000000000000000000000000000000000000000000000000000000
HEX: 8000000000000000
Zamiana na typ int, m = -1
BIN: 11111111111111111111111111111111
HEX: ffffffff
Zamiana na typ int, m = 0
BIN: 0
HEX: 0
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Przeanalizujmy fragmenty kodu źródłowego:

```
long n = Long.MAX_VALUE;
System.out.println("n = "+n);
System.out.println("BIN: "+Long.toBinaryString(n));
System.out.println("HEX: "+Long.toHexString(n));
```

Największą liczbą całkowitą dodatnią 64-bitową ze znakiem (typu `long`) jest $2^{63}-1$ (*n = 9223372036854775807*). W klasie opakowującej `Long` reprezentuje tę liczbę stała `MAX_VALUE`. Stosując statyczną metodę `toBinaryString` dla tej liczby, zobaczymy jej postać binarną — ciąg *63* cyfr *1*; sześćdziesiąty czwarty (najstarszy) bit ma wartość *0* i nie jest wyświetlany (pominięcie tzw. zer nieznaczących). W postaci szesnastkowej, uzyskanej metodą statyczną `toHexString`, otrzymujemy szesnaście cyfr — cyfrę *7* (binarnie *0111*) i piętnaście cyfr *f* (binarnie *1111*, dziesiętnie *15*).

```
long m = n+1;
System.out.println("m = "+m);
System.out.println("BIN: "+Long.toBinaryString(m));
System.out.println("HEX: "+Long.toHexString(m));
```

Po dodaniu liczby *1* otrzymamy (zgodnie z regułami działań na liczbach binarnych) ciąg bitów złożony z jedynki i *63* zer. Odpowiada to liczbie dziesiętnej -2^{63} (*–9223 372036854775808*). Najstarszy bit (*1*) jest w tym przypadku bitem znaku i oznacza liczbę ujemną (w kodzie uzupełnień do dwóch). W postaci szesnastkowej mamy cyfrę *8* (binarnie *1000*) i piętnaście cyfr *0* (binarnie *0000*).

```
Long max = new Long(n);
System.out.println("Zamiana na typ int, m = "+max.intValue());
System.out.println("BIN: "+Integer.toBinaryString(max.intValue()));
System.out.println("HEX: "+Integer.toHexString(max.intValue()));
```

Tworzymy obiekt `max` (reprezentujący liczbę `n`) klasy opakowującej `Long`. Stosując metodę `intValue`, otrzymamy liczbę typu `int` równą *–1* (w postaci binarnej *11111111111 11111111111111111111* — *32* bity *1*). Z liczby ośmiobajtowej (`long`) obcięto cztery starsze bajty i zostawiono resztę (cztery młodsze bajty) — liczbę 32-bitową typu `int`.

Podobny efekt można uzyskać, wykonując rzutowanie liczby typu `long` na typ `int`.

```
System.out.println("Zamiana na typ int, m = "+(int)m);
System.out.println("BIN: "+Integer.toBinaryString((int)m));
System.out.println("HEX: "+Integer.toHexString((int)m));
```

Przy wyświetlaniu liczb zostały pominięte zera nieznaczące.

Zmieniając wartość zmiennej `n` na minimalną oraz biorąc liczbę `m` o jeden mniejszą

```
(Long n = new Long(Long.MIN_VALUE); Long),
```

otrzymamy taki rezultat:

```
n = -9223372036854775808
BIN: 1000000000000000000000000000000000000000000000000000000000000000
HEX: 8000000000000000
m = 9223372036854775807
BIN: 1111111111111111111111111111111111111111111111111111111111111111
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
HEX: 7fffffffffffffff
Zamiana na typ int, m = 0
BIN: 0
HEX: 0
Zamiana na typ int, m = -1
BIN: 11111111111111111111111111111111
HEX: ffffffff
```

Analizę wyniku pozostawiamy Czytelnikowi.

Zadanie 8.2. MinMax.java, Z08_2.java

Liczba całkowita *n*-bitowa ze znakiem ma najmniejszą wartość -2^{n-1} (bit *1* i *n–1* bitów *0*) i największą wartość $2^{n-1}-1$ (*n* bitów *1*). W Javie typy proste `byte`, `short`, `int` i `long` są typami liczb całkowitych ze znakiem, o długości (odpowiednio do kolejności podanych nazw) *8*, *16*, *32* i *64* bitów. Nie musimy obliczać wartości granicznych według podanej wyżej zasady, gdyż klasy opakowujące `Byte`, `Short`, `Integer` i `Long` zawierają odpowiednie stałe `MIN_VALUE` i `MAX_VALUE`.

```
public class MinMax {
    public static void main(String args[]) {
        System.out.println("byte  <"+
            Byte.MIN_VALUE+", "+Byte.MAX_VALUE+">");
        System.out.println("short <"+Short.MIN_VALUE+", "+
            Short.MAX_VALUE+">");
        System.out.println("int   <"+Integer.MIN_VALUE+
            ", "+Integer.MAX_VALUE+">");
        System.out.println("long  <"+Long.MIN_VALUE+
            ", "+Long.MAX_VALUE+">");
    }
}
```

Po wykonaniu programu otrzymamy wynik:

```
byte  <-128, 127>
short <-32768, 32767>
int   <-2147483648, 2147483647>
long  <-9223372036854775808, 9223372036854775807>
```

Zadanie 8.3. MaxPositive.java, Z08_3.java

W rozwiązaniu zadania wykorzystamy stałe z klas opakowujących proste typy liczbowe: `Byte.MAX_VALUE`, `Short.MAX_VALUE`, `Integer.MAX_VALUE` i `Long.MAX_VALUE`, a także metody `toBinaryString` i `toHexString` — z klasy `Integer` dla typów `byte`, `short` i `int` oraz z klasy `Long` dla typu `long` (lub wszystkich wymienionych typów).

```
public class MaxPositive {
    public static void main(String args[]) {
        System.out.print("byte  "+
           Integer.toBinaryString(Byte.MAX_VALUE));
        System.out.println("\t"+Integer.toHexString(Byte.MAX_VALUE));
        System.out.print("short "+
           Integer.toBinaryString(Short.MAX_VALUE));
        System.out.println("\t"+Integer.toHexString(Short.MAX_VALUE));
        System.out.print("int   "+
           Integer.toBinaryString(Integer.MAX_VALUE));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("\t"+Integer.toHexString(Integer.MAX_VALUE));
        System.out.print("long  "+Long.toBinaryString(Long.MAX_VALUE));
        System.out.println("\t"+Long.toHexString(Long.MAX_VALUE));
    }
}
```

Zadanie 8.4. Kodowanie.java, Z08_4.java, Z08_4a.java

Po uruchomieniu aplikacji:

```
public class Kodowanie {
    public static void main(String[] args) {
        String napis = "kodowanie";
        System.out.println("Tekst: "+napis);
        System.out.print("Ciąg bajtów:");
        byte[] kod = napis.getBytes();
        for(byte n : kod)
             System.out.print(" "+n+",");
        System.out.println("\b.");
    }
}
```

uzyskamy następujący wynik:

```
Tekst: kodowanie
Ciąg bajtów: 107, 111, 100, 111, 119, 97, 110, 105, 101.
```

Zwróćmy uwagę na sposób wyświetlania poszczególnych elementów tablicy (w pętli): odstęp, liczba i przecinek. Przecinek po ostatniej liczbie kasujemy, wpisując znak `\b` (ang. *backspace*), a następnie stawiamy kropkę.

Jeśli skorzystamy z klasy `Arrays` (potrzebny będzie import klasy `java.util.Arrays`) opakowującej typy tablicowe, to możemy tablicę bajtów `kod` zamienić na łańcuch znaków w postaci `[107, 111, 100, 111, 119, 97, 110, 105, 101]` i wyświetlić go w konsoli. Użyjemy do tego celu statycznej metody `Arrays.toString()`, np.:

```
System.out.print("Ciąg bajtów: ");
System.out.println(Arrays.toString(napis.getBytes()));
```

Zadanie 8.5. Dekodowanie.java, Z08_5.java

Na podstawie ciągu bajtów tworzymy tablicę bajtów `byte[] kod = {115, 122, 121, 102, 114}`, wyświetlamy zawartość tablicy w konsoli (zob. drugi wariant rozwiązania zadania 8.4). Na podstawie tablicy bajtów (`kod`) tworzymy łańcuch znaków — klasa `String` posiada odpowiedni konstruktor (`String wynik = new String(kod)`).

```
import java.util.Arrays;
public class Dekodowanie {
    public static void main(String[] args) {
        byte[] kod = {115, 122, 121, 102, 114};
        System.out.println("Ciąg bajtów: "+Arrays.toString(kod));
        String wynik = new String(kod);
        System.out.println("Tekst: "+wynik);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 8.6. DodajCyfry.java, Z08_6a.java, Z08_6b.java

a) Liczba podana w postaci łańcucha znaków (cyfr) nie ma właściwie ograniczenia liczby cyfr powodowanego zakresem całkowitych typów liczbowych w Javie. Ciąg cyfr zamieniamy na ciąg (tablicę) odpowiadających im bajtów. Podczas sumowania od kolejnych bajtów odejmujemy *48* (kod *0*) i uzyskujemy wartość liczbową odpowiedniej cyfry.

```
public class DodajCyfry {
    public static void main(String[] args) {
        String liczba = "3784596320";
        byte[] kod = liczba.getBytes();
        int suma = 0;
        for(byte c: kod)
            suma += c-48;
        System.out.println("Liczba naturalna: "+liczba);
        System.out.println("Suma cyfr: "+suma);
    }
}
```

b) Jeśli liczba zapisana jest w pamięci w postaci numerycznej (proponujemy użyć typu o największym zakresie, czyli `long`), to zamieniamy ją na łańcuch znaków:

```
long n = 3784596320L;
String liczba = Long.toString(n);
```

i kontynuujemy obliczenia według podpunktu a). Należy zwrócić uwagę na sposób zainicjowania zmiennej `n` — liczby całkowite większe od `Integer.MAX_VALUE` (*2147483647*) są liczbami typu `long` i reprezentujące je literały muszą być zakończone literą `L` (można również użyć małej litery `l`, ale ten znak możemy pomylić z cyfrą `1`). Największą liczbą, dla której możemy to zadanie wykonać, jest `Long.MAX_VALUE`, czyli *9223372036854775807*.

Zadanie 8.7. Txt2Hex.java, Z08_7.java

Podobnie jak w rozwiązaniu zadania 8.4, zamienimy łańcuch znaków (`String napis = "szyfr"`) na ciąg bajtów zapisany w tablicy `kod` (`byte[] kod = napis.getBytes()`). Wyświetlając zawartość tablicy, zamienimy liczby (`n` typu `byte`) na ich reprezentację szesnastkową (`Integer.toHexString(n).toUpperCase()`).

```
public class Txt2Hex {
    public static void main(String[] args) {
        String napis = "szyfr";
        System.out.print(napis+" -> ");
        byte[] kod = napis.getBytes();
        for(byte n : kod)
            System.out.print(Integer.toHexString(n).toUpperCase());
    }
}
```

Zadanie 8.8. Hex2Txt.java, Z08_8.java

Łańcuch `hex` zawiera zakodowany ciąg znaków (`String hex = "737A796672"`). Każde dwie cyfry szesnastkowe (`bajt`) odpowiadają jednemu znakowi. Tworzymy tablicę

Ebookpoint.pl kopia dla: pawher@o2.pl

bajtów o rozmiarze dwa razy mniejszym od rozmiaru łańcucha `hex` (`byte[] tmp = new byte[hex.length()/2]`). W pętli `for` z łańcucha `hex` pobieramy po dwie cyfry szesnastkowe (`hex.substring(2*i,2*i+2)`), zamieniamy je na liczby i przechowujemy w tablicy `tmp`. Na koniec zamieniamy tablicę bajtów na łańcuch znaków (`new String(tmp)`).

```
public class Hex2Txt {
    public static void main(String[] args) {
        String hex = "737A796672";
        System.out.print(hex+" -> ");
        byte[] tmp = new byte[hex.length()/2];
        for(int i = 0; i < tmp.length; ++i)
            tmp[i] = Byte.parseByte(hex.substring(2*i,2*i+2), 16);
        System.out.println(new String(tmp));
    }
}
```

9. Typy liczb zmiennoprzecinkowych

Zadanie 9.1. DemoDouble.java, Z09_1.java

Zobaczmy wyniki pracy programu:

```
Minimalna wartość dodatnia: 4.9E-324
BIN: 1
HEX: 0x0.0000000000001p-1022
Liczba przeciwna: -4.9E-324
BIN: 1000000000000000000000000000000000000000000000000000000000000001
HEX: -0x0.0000000000001p-1022
```

Stała `Double.MIN_VALUE` ma wartość najmniejszej liczby dodatniej możliwej do przedstawienia w postaci liczby zmiennoprzecinkowej podwójnej precyzji (typu `double`), czyli wartość `4.9E-324` ($4{,}9 \cdot 10^{-324}$). Jej reprezentacja binarna ma postać (64 bity, w konsoli zera nieznaczące zostały pominięte):

```
0000000000000000000000000000000000000000000000000000000000000001
```

W skład tej reprezentacji wchodzą: `0` (0 oznacza liczbę dodatnią), 11 bitów cechy `000 00000000` i 52 bity mantysy `0000000000000000000000000000000000000000000000000001`.

Ponieważ wszystkie bity cechy są zerami, więc mantysa jest zdenormalizowana (bez bitu całkowitego `1`) i ma postać binarną `0.0000000000000000000000000000000000000000 00000000000001` (cyfra `1` na pięćdziesiątej drugiej pozycji po przecinku), a to odpowiada liczbie dziesiętnej 2^{-52}. Mnożąc mantysę przez liczbę 2^{1-1023}, uzyskamy wartość $2^{-52} \cdot 2^{1-1023} = 2^{-1074} = 4{,}9 \cdot 10^{-324}$.

Zapis binarny liczby przeciwnej różni się jedynie bitem znaku: `1` oznacza liczbę ujemną.

W programie zastosowano następujące elementy statyczne z klas `Double` i `Long`:

- `Double.MIN_VALUE` — stała o wartości najmniejszej liczby dodatniej typu `double`,

Ebookpoint.pl kopia dla: pawher@o2.pl

- `long Double.doubleToLongBits(double)` — metoda zamieniająca liczbę zmiennoprzecinkową typu `double` na liczbę całkowitą typu `long` o identycznej reprezentacji binarnej,
- `String Long.toBinaryString(long)` — metoda zamieniająca liczbę całkowitą typu `long` na łańcuch zawierający reprezentację binarną tej liczby (z pominięciem zer nieznaczących),
- `String Double.toHexString(double)` — metoda zamieniająca liczbę zmiennoprzecinkową typu `double` na łańcuch znaków heksadecymalnych reprezentujących tę liczbę; np. w ciągu znaków `0x0.0000000000001p-1022` kolejne elementy oznaczają:

  `0x` — liczba jest zapisana w *kodzie szesnastkowym*;

  `0.0000000000001` — zdenormalizowana *mantysa* — cyfra jedności `0`, 13 cyfr szesnastkowych po przecinku (13·4 = 52 bity mantysy);

  `p-1022` — *wykładnik* (–1022) zapisany w kodzie dziesiątkowym (podstawą potęgi jest 2).

  Zapis ten reprezentuje liczbę zmiennoprzecinkową o wartości $mantysa \cdot 2^{1022}$.

Zadanie 9.2. DoubleBin.java, Z09_2.java

Łańcuch `zero` zawierający *63* znaki *0* przyda się do uzupełniania zer nieznaczących do *64* bitów w liczbach dodatnich, dla których metoda `toBinaryString` zera nieznaczące pomija. Liczby lub symbole, których postać binarną chcemy poznać, umieszczamy w tablicy `double[] a = {...}`. W pętli pobieramy elementy (`for(double x : a)...`) tej tablicy, wyświetlamy ich postać zmiennoprzecinkową w konsoli (`System.out.println ("x = "+x)`), zamieniamy liczbę `x` typu `double` na liczbę typu `long`, mającą identyczną reprezentację binarną (`Double.doubleToLongBits(x)`), a tę liczbę typu `long` przedstawiamy w postaci binarnej (`String tmp = Long.toBinaryString(...)`) — łańcuch `tmp`. Z łańcucha `tmp` tworzymy łańcuch `bin` o długości *64* znaków (słowo 64-bitowe), dodając zera (nieznaczące) na początku łańcucha.

Z łańcucha `bin` wybieramy bit znaku (`bin.charAt(0)`), cechę (`bin.substring(1, 12)`) i mantysę (`bin.substring(12)`), przedstawiając w ten sposób binarną postać liczby lub symbolu zmiennoprzecinkowego.

```
public class DoubleBin {
    static String zero =
      "0000000000000000000000000000000000000000000000000000000000000000";
    public static void main(String[] args) {
        double[] a = {0.25, 0.5, 1.0, 2.0, 512.0}; /* podpunkt a) */
        for(double x : a) {
            System.out.println("x = "+x);
            String tmp = Long.toBinaryString(Double.doubleToLongBits(x));
            String bin = zero.substring(0, 64-tmp.length())+tmp;
            System.out.println("BIN: "+bin);
            System.out.println("Bit znaku: "+bin.charAt(0));
            System.out.println("Cecha:     "+bin.substring(1, 12));
            System.out.println("Mantysa:   "+bin.substring(12));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

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

a) Wybrane potęgi liczby: *0,25*; *0,5*; *1,0*; *2,0*; *512,0*.

```
x = 0.25
BIN: 00111111110100000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     01111111101
Mantysa:   0000000000000000000000000000000000000000000000000000

x = 0.5
BIN: 00111111111000000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     01111111110
Mantysa:   0000000000000000000000000000000000000000000000000000

x = 1.0
BIN: 00111111111100000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     01111111111
Mantysa:   0000000000000000000000000000000000000000000000000000

x = 2.0
BIN: 01000000000000000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     10000000000
Mantysa:   0000000000000000000000000000000000000000000000000000

x = 512.0
BIN: 01000000100000000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     10000001000
Mantysa:   0000000000000000000000000000000000000000000000000000
```

Wszystkie liczby są dodatnie (bit znaku *0*). Ich cechy są różne od zera, więc mantysy są znormalizowane i równe (binarnie 1,0000…) 1. Cechy kolejnych liczb są równe binarnie 01111111101, 01111111110, 01111111111, 10000000000 i 10000001000, co odpowiada liczbom dziesiętnym 1021, 1022, 1023, 1024 i 1032. Po uwzględnieniu przesunięcia –1023 otrzymamy wykładniki –2, –1, 0, 1 i 9. Ponieważ mantysa jest równa 1 dla wszystkich liczb z podanego zestawu, to wartości tych liczb są równe $2^{-2} = 0{,}25$; $2^{-1} = 0{,}5$; $2^0 = 1$; $2^1 = 2$ i $2^9 = 512$.

b) Podane liczby są potęgami liczby *10*.

```
double[] a = {0.01, 0.1, 1.0, 10.0, 100.0};
```

W postaci binarnej nie dostrzeżemy żadnego związku pomiędzy tymi liczbami.

c) Podane liczby są liczbami dziesiętnymi zapisanymi przy użyciu tych samych cyfr i różnią się położeniem przecinka dziesiętnego.

```
double[] a = {1.367, 1367, 1.367e-12, 1.367e12, 13.67, -13.67};
```

Ebookpoint.pl kopia dla: pawher@o2.pl

W postaci binarnej nie dostrzeżemy żadnych podobieństw, z wyjątkiem ostatnich dwu liczb, które są liczbami przeciwnymi (różnią się bitem znaku).

d) Symbole specjalne (*NaN* i $\pm\infty$).

```
double[] a = {Double.NaN, Double.NEGATIVE_INFINITY, Double.POSITIVE_INFINITY};
```

Wszystkie symbole specjalne mają cechę złożoną z samych bitów `1`. Mantysa znaku `NaN` (ang. *Not a Number*) składa się z bitu `1` i 51 bitów `0`.

```
x = NaN
BIN: 0111111111111000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     11111111111
Mantysa:   1000000000000000000000000000000000000000000000000000
```

Symbole nieskończoności mają mantysy złożone z bitów `0`, a bit znaku decyduje o znaku wartości.

```
x = -Infinity
BIN: 1111111111110000000000000000000000000000000000000000000000000000
Bit znaku: 1
Cecha:     11111111111
Mantysa:   0000000000000000000000000000000000000000000000000000

x = Infinity
BIN: 0111111111110000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     11111111111
Mantysa:   0000000000000000000000000000000000000000000000000000
```

e) Dwie reprezentacje zera (dodatnia i ujemna) oraz dodatnie wartości ekstremalne.

```
double[] a = {0.0, -0.0, Double.MIN_VALUE, Double.MAX_VALUE,
2.2250738585072014e-308};
```

Cecha i mantysa składają się z samych bitów *0*. To reprezentacja liczby `0.0`. Jedynie bit znaku może przyjmować wartość `0` lub `1`. Mamy zatem dwie reprezentacje binarne zera w liczbach zmiennoprzecinkowych — w informatyce używamy czasem wyrażeń „zero dodatnie” i „zero ujemne”.

```
x = 0.0
BIN: 0000000000000000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     00000000000
Mantysa:   0000000000000000000000000000000000000000000000000000
x = -0.0
BIN: 1000000000000000000000000000000000000000000000000000000000000000
Bit znaku: 1
Cecha:     00000000000
Mantysa:   0000000000000000000000000000000000000000000000000000
```

Tę wartość (najmniejszą możliwą do zapisania liczbę dodatnią — stałą `Double.MIN_VALUE`) omówiono dokładnie w rozwiązaniu zadania 9.1 (mantysa tej liczby nie jest znormalizowana).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
x = 4.9E-324
BIN: 0000000000000000000000000000000000000000000000000000000000000001
Bit znaku: 0
Cecha:     00000000000
Mantysa:   0000000000000000000000000000000000000000000000000001
```

Największa możliwa do zapisania liczba zmiennoprzecinkowa (stała `Double.MAX_VALUE`) ma cechę `11111111110` (największą, jaką można zapisać na *11* bitach, bo cecha `11111111111` jest zarezerwowana dla symboli specjalnych) i mantysę (znormalizowaną) złożoną z samych bitów *1*.

```
x = 1.7976931348623157E308
BIN: 0111111111101111111111111111111111111111111111111111111111111111
Bit znaku: 0
Cecha:     11111111110
Mantysa:   1111111111111111111111111111111111111111111111111111
```

Binarna mantysa `1.1111111111111111111111111111111111111111111111111111` po przeliczeniu na układ dziesiątkowy ma wartość $2–2^{–52}$, cecha `11111111110` ma wartość *2046*, więc wartość dziesiętna tej liczby wynosi $(2–2^{–52})·2^{2046–1023} = (2–2^{–52})·2^{1023} = 1,7976931348623157·10^{308}$.

Kolejna liczba jest najmniejszą liczbą dodatnią o znormalizowanej mantysie $1,0·2^{1–1023} = 2^{–1022} = 2,2250738585072014·10^{-308}$.

```
x = 2.2250738585072014E-308
BIN: 0000000000010000000000000000000000000000000000000000000000000000
Bit znaku: 0
Cecha:     00000000001
Mantysa:   0000000000000000000000000000000000000000000000000000
```

Zadanie 9.3. Bin2Float.java, Z09_3.java

Ciąg bitów o długości nieprzekraczającej *32* znaków zapisujemy w łańcuchu `bin`. Obliczmy wartość całkowitą `n` odpowiadającą podanemu łańcuchowi cyfr binarnych (`int n = Integer.parseInt(bin, 2)`). Liczbę całkowitą typu `int` zamieniamy na liczbę zmiennoprzecinkową typu `float` o identycznej reprezentacji binarnej (`float x = Float.intBitsToFloat(n)`).

```
public class Bin2Float {
    public static void main(String[] args) {
        String bin ="11111000001111000011100011001O";
        int n = Integer.parseInt(bin, 2);
        float x = Float.intBitsToFloat(n);
        System.out.println("bin:   "+bin);
        System.out.println("int:   "+n);
        System.out.println("float: "+x);
    }
}
```

Ciąg bitów *11111000001111000011100011001O* reprezentuje liczbę zmiennoprzecinkową (`float`) *0,13970259*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 9.4. Bin2Double.java, Z09_4.java

Ciąg bitów o długości nieprzekraczającej *64* znaków zapisujemy w łańcuchu `bin`. Obliczmy wartość całkowitą `n` (*64*-bitową, typu `long`) odpowiadającą podanemu łańcuchowi cyfr binarnych (`long n = Integer.parseLong(bin, 2)`). Liczbę całkowitą typu `long` zamieniamy na liczbę zmiennoprzecinkową typu `double` o identycznej reprezentacji binarnej (`double x = Double.longBitsToDouble(n)`).

```
public class Bin2Double {
    public static void main(String[] args) {
        String bin ="11111000001111000011100011 0010";
        long n = Long.parseLong(bin, 2);
        double x = Double.longBitsToDouble(n);
        System.out.println("bin:    "+bin);
        System.out.println("long:   "+n);
        System.out.println("double: "+x);
    }
}
```

Ciąg bitów *11111000001111000011100011 0010* reprezentuje liczbę zmiennoprzecinkową (`double`) *5,144083374E-315* (porównaj ten wynik z wynikiem uzyskanym w zadaniu 9.3).

Zadanie 9.5. Bin2Double.java, Z09_5.java

W symbolach specjalnych cecha ma wszystkie bity równe `1` (w ciągach bitów symbole te zostały wyróżnione przez pogrubienie). Oto reprezentacje binarne trzech symboli specjalnych):

- nieskończoność ujemna ($-\infty$) **11111111**00000000000000000000000,
- nieskończoność dodatnia ($+\infty$) **01111111**00000000000000000000000,
- nie liczba (`NaN`) **01111111**11111111111111111111111 (co najmniej najstarszy bit mantysy musi być równy `1`).

```
public class TestBinFloat {
    public static void main(String[] args) {
        float x = Float.intBitsToFloat(Integer.parseInt(args[0], 2));
        System.out.println(args[0]+" -> "+x);
    }
}
```

Kod wygląda na poprawny, ale niestety kłopoty sprawia metoda `Integer.parseInt (String, int)`, która nie akceptuje (zgodnie z dokumentacją) ciągów cyfr binarnych 32-bitowych, mających najstarszy bit `1` (liczby całkowite ujemne w kodzie uzupełnień do 2), natomiast dopuszcza zastosowanie znaku – (minus) przed ciągiem cyfr binarnych, nie dłuższym niż 31 bitów.

Aby wyznaczyć liczbę typu `float` mającą reprezentację binarną 11111111000000000 00000000000000, powinniśmy zanegować w tym ciągu wszystkie bity 0**00000000**1111 111111111111111111, a następnie dodać `1` (dodawanie zgodnie z regułami dodawania liczb w systemie binarnym). Otrzymamy w ten sposób ciąg cyfr 0**00000001**000000000 00000000000000, przed którym stawiamy znak minus (-0**00000001**00000000000000000

Ebookpoint.pl kopia dla: pawher@o2.pl

000000). Zera nieznaczące można pominąć — ich obecność jednak nie wpływa na wynik (w tym przypadku -Infinity, czyli nieskończoność ujemna).

Rozważana sytuacja nie obejmuje najmniejszej liczby typu int (binarnie 10000000000000000000000000000000), której w typie float odpowiada wartość -0.0 (zero ujemne!). Możemy o tym się przekonać, wykorzystując następujący wiersz kodu:

```
System.out.println(Float.intBitsToFloat(Integer.MIN_VALUE));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 6.

Rozwiązania zadań

10. Wyświetlanie sformatowanych wyników w konsoli. Stałe i metody z klasy Math

Zadanie 10.1. Z10_1.java

Najprostszym rozwiązaniem jest użycie metody `printf()` wprowadzonej w *J2SE ver. 5.0 (1.5)*. Umieszczony w łańcuchu formatującym `"4/7 = %.5f\n"` ciąg znaków `%.5f` jest specyfikatorem określającym, że parametr `x` podstawiony w miejsce specyfikatora ma być zinterpretowany jako liczba zmiennoprzecinkowa (`f`) i wyświetlona z precyzją 5 miejsc po przecinku (`.5`). Symbol `%` sygnalizuje, że występujące po nim znaki tworzą specyfikator.

```
public class Z10_1 {
    public static void main(String[] args) {
        double x = (double)4/7;
        System.out.printf("4/7 = %.5f\n", x);
    }
}
```

Poznane wcześniej metody `print()` i `println()` również pozwalają wyświetlić liczby z określoną precyzją, pod warunkiem że odpowiednio zaokrąglimy wynik. Pomnóżmy wartość zmiennej `x` przez `100000` (przesunięcie przecinka o 5 miejsc w prawo), dodajmy `0.5` i obetnijmy część ułamkową liczby (konwersja liczby zmiennoprzecinkowej na liczbę całkowitą przez rzutowanie), a następnie podzielmy wynik przez `100000.0` (dzielimy liczbę całkowitą przez liczbę zmiennoprzecinkową, aby uzyskać wynik zmiennoprzecinkowy). Otrzymany rezultat jest poprawnym zaokrągleniem wartości `x` do 5 miejsc po przecinku.

```
double y = (int)(100000*x+0.5)/100000.0;
System.out.println("4/7 = "+y);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Możemy zastosować zaokrąglenie przy użyciu metody `round()`.

```
y = Math.round(100000*x+0.5)/100000.0;
```

Podobne obliczenia możemy zrealizować z zastosowaniem obiektu klasy `Double`.

```
y = new Double(100000*x+0.5).intValue()/100000.0;
```

Inną możliwość stwarza metoda `format()` klasy `String`:

```
System.out.println(String.format("4/7 = %.5f", x));
```


Uwaga

Zauważmy podobieństwo pomiędzy pierwszym i ostatnim przypadkiem (sposób tworzenia łańcucha znaków). Uzyskany wynik jest zgodny z ustawieniami lokalnymi komputera: *0,57143* (przecinek dziesiętny). W pozostałych przypadkach metoda `println()` wyświetla liczby dziesiętne z kropką.

Zadanie 10.2. Z10_2.java

W pakiecie `java.lang` zdefiniowano klasę `Math` zawierającą wartości przybliżone stałych *e* i π oraz różne funkcje matematyczne. Dostęp do stałych lub funkcji otrzymujemy, podając nazwę klasy, kropkę i nazwę stałej lub funkcji, np. `Math.E`, `Math.PI` (nazwy stałych piszemy zwyczajowo wielkimi literami[1]) lub `Math.sqrt(x)` — `x` jest argumentem funkcji `sqrt` (ang. *square root* — pierwiastek kwadratowy).

```
public class Z10_2 {
    public static void main(String[] args) {
        System.out.printf("Liczba  e = %15.10f\n", Math.E);
        System.out.printf("Liczba pi = %15.10f\n", Math.PI);
        System.out.printf("Liczba fi = %15.10f\n", (1+Math.sqrt(5))/2);
    }
}
```

Możemy *statycznie importować* klasę `Math` i w zapisie wzorów pominąć nazwę klasy.

```
import static java.lang.Math.*;
public class Z10_2a {
    public static void main(String[] args) {
        System.out.printf("Liczba  e = %15.10f\n", E);
        System.out.printf("Liczba pi = %15.10f\n", PI);
        System.out.printf("Liczba fi = %15.10f\n", (1+sqrt(5))/2);
    }
}
```

Zadanie 10.3. Z10_3.java

W klasie `Math` oprócz funkcji `sqrt()` obliczającej pierwiastek kwadratowy zdefiniowano funkcję `cbrt()` obliczającą pierwiastek trzeciego stopnia (ang. *cube root* — pierwiastek sześcienny). W pętli typu `for each` zmienna `n` przyjmuje kolejno wartości zapisane w tablicy `dane`. Dla każdej wartości zmiennej `n` metoda `printf()` wyświetla w konsoli zgodnie z łańcuchem formatującym:

[1] Standard kodowania zaleca pisanie nazw stałych wielkimi literami; ostateczna decyzja w tej sprawie należy do programującego.

Ebookpoint.pl kopia dla: pawher@o2.pl

- `%3d` — na polu o szerokości 3 znaków liczbę całkowitą `n`;
- `%15.8f` — na polu o szerokości 15 znaków liczbę zmiennoprzecinkową (wartość wyrażenia `sqrt(n)`) z dokładnością do 8 miejsc po przecinku;
- `%15.8f` — na polu o szerokości 15 znaków liczbę zmiennoprzecinkową (wartość wyrażenia `cbrt(n)`) z dokładnością do 8 miejsc po przecinku;
- `\n` — znak końca linii; kursor przechodzi na początek nowego wiersza.

```
import static java.lang.Math.*;
public class Z10_3 {
    public static void main(String[] args) {
        int[] dane = {2, 3, 5, 7, 9, 11, 13, 17};
        for(int n : dane)
            System.out.printf("%3d%15.8f%15.8f\n", n, sqrt(n), cbrt(n));
    }
}
```

Zadanie 10.4. Z10_4.java

Funkcja `pow` (ang. *power* — potęga) z klasy `Math` posiada dwa parametry. Pierwszy jest podstawą, a drugi wykładnikiem obliczanej potęgi. Wzór $x = \sqrt[n]{5} = 5^{\frac{1}{n}}$ możemy zapisać w postaci `x = Math.pow(5, 1.0/n)`.

```
public class Z10_4 {
    public static void main(String[] args) {
        int[] dane = {2, 3, 4, 5, 6, 7, 8, 9, 10};
        for(int n : dane) {
            double x = Math.pow(5, 1.0/n);
            System.out.printf("Pierwiastek %2d stopnia z 5: %f\n", n, x);
        }
    }
}
```

Precyzja 6 miejsc po przecinku jest domyślnie ustawiana dla metody `printf()`, więc w tym zadaniu użyto specyfikatora `%f` zamiast `%.6f`. Zwróćmy również uwagę na sposób obliczania wykładnika — wyrażenie `1.0/n` daje nam zmiennoprzecinkową wartość ułamka (`1/n` — dzielenie całkowite dałoby wykładnik równy `0`).

Zadanie 10.5. Z10_5.java

Najpierw wyświetlamy wiersz nagłówka, rezerwując na pierwszą kolumnę 4 znaki (rozmiar słowa `Znak`) i na pozostałe po 10 znaków (specyfikator `%10s` — pole o szerokości 10 znaków dla łańcucha znaków, `s` — string).

Łańcuch znaków `"ABCDEFGHIJKLMNOPQRSTUVWXYZ"` zamieniamy, stosując metodę `toCharArray()`, na tablicę znaków `znaki`. W pętli `for each` przeglądamy zawartość tablicy znaków i wyświetlamy w konsoli znak `z` oraz kod tego znaku `(int)z` (rzutowanie wartości typu `char` na typ `int`). W specyfikatorach występują symbole `%1$` i `%2$`. W miejsce `%1$` wstawiana jest wartość pierwszego parametru (znak `z`), a w miejsce `%2$` — wartość drugiego parametru (kod znaku `(int)z`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Z10_5 {
    public static void main(String[] args) {
        System.out.printf("Znak%10s%10s%10s\n", "OCT", "DEC", "HEX");
        char[] znaki = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
        for(char z : znaki) {
            System.out.printf("%1$3c %2$10o%2$10d%2$10X\n", z, (int)z);
        }
    }
}
```

Przeanalizujmy specyfikatory w łańcuchu formatującym:

- `%1$3c` — zmienna typu `char` (pierwszy parametr) jest wyświetlana na polu szerokości 3 znaków;
- `%2$10o` — liczba całkowita (drugi parametr) wyświetlana jest w systemie ósemkowym (oktalnym) na polu o szerokości 10 znaków;
- `%2$10d` — liczba całkowita (drugi parametr) wyświetlana jest w systemie dziesiętnym (decymalnym) na polu o szerokości 10 znaków;
- `%2$10X` — liczba całkowita (drugi parametr) wyświetlana jest w systemie szesnastkowym (heksadecymalnym) na polu o szerokości 10 znaków. Cyfry szesnastkowe (ABCDEF) wyświetlane są wielkimi literami.

Zadanie 10.6. Z10_6.java

Metoda (funkcja) `toDegrees()` z klasy `Math` zamienia miarę kąta podaną w radianach na miarę wyrażoną w stopniach. Symbol stopnia w tekście uzyskujemy, wpisując znak `'\u00B0'`.

```
public class Z10_6 {
    public static void main(String[] args) {
        double alfa = Math.toDegrees(1); // 1 radian w stopniach
        System.out.println("1 rad = "+alfa+"\u00B0");
        int st, min, sek;
        st = (int)alfa;
        min = (int)((alfa-st)*60+0.5);
        System.out.printf("1 rad = %d\u00B0%02d\'\n", st, min);
        min = (int)((alfa-st)*60);
        sek = (int)((alfa-st-min/60.0)*3600+0.5);
        System.out.printf("1 rad = %d\u00B0%02d\'%02d\"\n", st, min, sek);
    }
}
```

Aby zrozumieć algorytm zamiany miary stopniowej kąta wyrażonej ułamkiem dziesiętnym na stopnie (wyrażone liczbą całkowitą), minuty i sekundy kątowe, prześledźmy rachunek dla kąta o mierze 1 radiana:

- `1 rad = 57.29577951308232°` — wynik działania metody `Math.toDegrees(1);`.
- `1 rad = 57°18'` — liczba stopni jest częścią całkowitą liczby *57.29577951308232* (`st = (int)alfa;`), a liczba minut jest częścią całkowitą iloczynu *0,29577951308232·60 = 17,7467707849392*, po zaokrągleniu (`min = (int)((alfa-st)*60+0.5);`).

Ebookpoint.pl kopia dla: pawher@o2.pl

- `1 rad = 57°17'45"` — tym razem nie zaokrąglamy liczby minut, ale bierzemy tylko część całkowitą (`min = (int)((alfa-st)*60);`). Liczba sekund jest równa iloczynowi *0,7467707849392·60 = 44,806247096352* zaokrąglonemu do części całkowitej (`sek = (int)((alfa-st-min/60.0)*3600+0.5);`).

Zadanie 10.7. Z10_7.java

Metoda (funkcja) `toRadians()` z klasy `Math` zamienia miarę kąta podaną w stopniach na miarę w radianach. Obliczenia wykonamy dla liczb *1* (*1°*), $\frac{1}{60}$ (*1′*) i $\frac{1}{3600}$ (*1″*).

```
public class Z10_7 {
    public static void main(String[] args) {
        double rad = Math.toRadians(1.0);
        System.out.printf("1\u00B0 = %.15f rad\n", rad);
        rad = Math.toRadians(1.0/60);
        System.out.printf("1\' = %.15f rad\n", rad);
        rad = Math.toRadians(1.0/3600);
        System.out.printf("1\" = %.15f rad\n", rad);
    }
}
```

Zadanie 10.8. Z10_8.java

W trójkącie egipskim o bokach a = 3, b = 4 i c = 5 iloraz $\frac{a}{c} = \frac{3}{5}$ jest jednocześnie sinusem jednego kąta ostrego i cosinusem drugiego kąta. Stosując funkcje odwrotne (dostępne w klasie `Math`), możemy wyznaczyć miary kątów ostrych wyrażone w radianach: `alfa = Math.asin(a/c)` i `beta = Math.acos(a/c)`. Według algorytmu opisanego w rozwiązaniu zadania 10.7 możemy zamienić miary w radianach na miarę w stopniach, stopniach i minutach oraz stopniach, minutach i sekundach i sformułować odpowiedzi.

```
public class Z10_8 {
    public static void main(String[] args) {
        double a = 3.0, b = 4.0, c = 5.0;
        double alfa, beta;
        int st, min; // do podpunktu c) i d)
        int sek;     // do podpunktu c)
        alfa = Math.asin(a/c);
        beta = Math.acos(a/c);
        /* odpowiedź a) */
        System.out.printf("alfa = %.4f rad\nbeta = %.4f rad", alfa,
            beta);
        /* odpowiedź b) */
        System.out.println();
        alfa = Math.toDegrees(alfa);
        beta = Math.toDegrees(beta);
        System.out.printf("alfa = %.1f\u00B0\nbeta = %.1f\u00B0", alfa,
            beta);
        /* odpowiedź c) */
        System.out.println();
        st = (int)alfa;
        min = (int)((alfa-st)*60+0.5);
        System.out.printf("alfa = %d\u00B0%02d\'\n", st, min);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        st = (int)beta;
        min = (int)((beta-st)*60+0.5);
        System.out.printf("beta = %d\u00B0%02d\'\n", st, min);
        /* odpowiedź d) */
        System.out.println();
        st = (int)alfa;
        min = (int)((alfa-st)*60);
        sek = (int)((alfa-st-min/60.0)*3600+0.5);
        System.out.printf("alfa = %d\u00B0%02d\'%02d\"\n", st, min, sek);
        st = (int)beta;
        min = (int)((beta-st)*60);
        sek = (int)((beta-st-min/60.0)*3600+0.5);
        System.out.printf("beta = %d\u00B0%02d\'%02d\"\n", st, min, sek);
    }
}
```

11. Wczytywanie danych — klasa Scanner

Zadanie 11.1. Z11_1.java

Zaczynamy od utworzenia obiektu `input` klasy `Scanner` odczytującego dane ze standardowego wejścia `System.in`. Zgodnie z treścią zadania dane wejściowe mogą być ułamkami dziesiętnymi, więc użyjemy zmiennej typu `float` (liczby zmiennoprzecinkowej pojedynczej precyzji) lub `double` (liczby zmiennoprzecinkowej podwójnej precyzji). Dla każdego typu danych mamy odrębną metodę wczytującą dane ze strumienia:

```
float t = input.nextFloat();
```

lub:

```
double t = input.nextDouble();
```

Dane wprowadzamy w postaci liczby całkowitej lub ułamka dziesiętnego (z przecinkiem). Dopuszczalny jest również format naukowy, np. `2,35e2` ($2{,}35 \cdot 10^2$).

```
import java.util.Scanner;
public class Z11_1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj temperaturę w \u00B0C, t = ");
        double t = input.nextDouble();
        System.out.printf("%.1f\u00B0C = %.1f\u00B0F\n", t, t*1.8+32);
        input.close();
    }
}
```

Dane wejściowe i wynik są wyświetlane z dokładnością do jednego miejsca po przecinku (specyfikator w łańcuchu formatującym `%.1f`). Symbol ° (stopnia) wprowadzany jest jako znak `\u00B0` (Unicode). Na koniec nie zapomnijmy o zamknięciu skanera `input.close()`.

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Ze strumienia danych możemy pobrać kolejny *token* (łańcuch znaków do najbliższego znaku białego — odstępu, tabulatora lub końca wiersza):

```
String s = input.next();
```

i zamienić go na liczbę zmiennoprzecinkową, stosując odpowiednią metodę statyczną klasy `Float` lub `Double`:

```
float t = Float.parseFloat(s);
```

lub:

```
float t = Float.valueOf(s);
```

lub:

```
double t = Double.parseDouble(s);
```

lub:

```
double t = Double.valueOf(s);
```

Nie jest to rozwiązanie korzystne — dane liczbowe będziemy musieli wprowadzać z kropką dziesiętną (konwencja angielska), a wyniki będą wyświetlane z przecinkiem. Lepiej używać zlokalizowanych metod `nextFloat()` lub `nextDouble()`.

Nie jest konieczne tworzenie obiektu `s` klasy `String`, możemy po prostu zapisać w jednym wierszu `float t = Float.parseFloat(input.next())`.

Zadanie 11.2. Z11_2.java

Wzór *°F = °C·1,8+32* przekształcamy na postać *°C = (°F-32)/1,8*. Temperatura wejściowa wyrażona jest liczbą całkowitą (zgodnie z treścią zadania), więc wartość pobieramy ze strumienia, stosując metodę `input.nextInt()`.

```
import java.util.Scanner;
public class Z11_2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj temperaturę w \u00B0F, t = ");
        int t = input.nextInt();
        System.out.printf("%d\u00B0F = %.1f\u00B0C\n", t, (t-32)/1.8);
        input.close();
    }
}
```

Ze strumienia danych możemy pobrać kolejny token:

```
String s = input.next();
```

i zamienić go na liczbę całkowitą, stosując wybraną metodę statyczną klasy `Integer`:

```
int t = Integer.parseInt(s);
```

lub:

```
int t = Integer.decode(s);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

lub:

```
int t = Integer.valueOf(s);
```

Możemy ten zapis skrócić, eliminując obiekt `s`, np.:

```
int t = Integer.valueOf(input.next());
```

Zadanie 11.3a. Z11_3a.java

Długość przeciwprostokątnej w trójkącie prostokątnym obliczymy ze wzoru $c = \sqrt{a^2 + b^2}$ wynikającego z twierdzenia Pitagorasa. Do dyspozycji w klasie `Math` mamy metodę `sqrt()` (pierwiastek kwadratowy) i metodę `pow()` (potęgowanie):

```
double c = Math.sqrt(Math.pow(a, 2)+Math.pow(b, 2));
```

Dwukrotne wywołanie metody `pow()` w tej sytuacji jest nieekonomiczne. Obliczanie kwadratu liczby wykonamy szybciej, stosując mnożenie:

```
double c = Math.sqrt(a*a+b*b);
```

Wprowadzanie danych z klawiatury i wyświetlanie wyniku nie sprawi problemu — należy jednak zauważyć, że program nie kontroluje poprawności danych.

```
import java.util.Scanner;
public class Z11_3a {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj długość przyprostokątnej, a = ");
        double a = input.nextDouble();
        System.out.print("Podaj długość przyprostokątnej, b = ");
        double b = input.nextDouble();
        double c = Math.sqrt(a*a+b*b);
        System.out.printf("Długość przeciwprostokątnej c = %.3f\n", c);
        input.close();
    }
}
```

Zadanie 11.3b. Z11_3b.java

Zadanie to jest typowym przykładem zastosowania funkcji trygonometrycznych (kąta ostrego) do obliczania elementów trójkąta prostokątnego (tzw. rozwiązywania trójkątów).

Przyjmując standardowe oznaczenia, obliczymy: $\sin\alpha = \frac{a}{c}$, $c \cdot \sin\alpha = a$, $c = \frac{a}{\sin\alpha}$.

Należy pamiętać, że funkcje trygonometryczne dostępne w klasie `Math` (`sin`, `cos` i `tan` — tangens) wymagają podania argumentu (miary kąta) w radianach. Do konwersji miary kąta możemy użyć metody `toRadians()` z klasy `Math`.

```
import java.util.Scanner;
public class Z11_3b {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj długość przyprostokątnej, a = ");
        double a = input.nextDouble();
        System.out.print("Podaj miarę kąta ostrego (°), alfa = ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        double alfa = input.nextDouble();
        double c = a/Math.sin(Math.toRadians(alfa));
        System.out.printf("Długość przeciwprostokątnej c = %.3f\n", c);
        input.close();
    }
}
```

Zadanie 11.4. Z11_4.java

Liczba bitów została ograniczona do *31* ze względu na zastosowanie typu zmiennych `int` i klasy `Integer`. Można liczbę bitów zwiększyć do *63*, stosując zmienne typu `long` i klasę opakowującą `Long`. Wykorzystując metodę `next()`, odczytujemy ze skanera kolejny token i następnie zamieniamy go na liczbę całkowitą. Metoda `parseInt()` z klasy `Integer` umożliwia określenie podstawy systemu liczbowego (drugi parametr). Po odkodowaniu (z systemu binarnego) dwóch liczb `a` i `b` możemy obliczyć ich sumę `a+b`, którą następnie zamieniamy na ciąg znaków reprezentujący jej postać binarną `Integer.toBinaryString(a+b)`.

```
import java.util.Scanner;
public class Z11_4 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Podaj dwie liczby w systemie binarnym (do 31 bitów)");
        System.out.print("a = ");
        String str = input.next();
        int a = Integer.parseInt(str, 2);
        System.out.print("b = ");
        str = input.next();
        int b = Integer.parseInt(str, 2);
        str = Integer.toBinaryString(a+b);
        System.out.println("a+b = "+str);
        input.close();
    }
}
```

Możemy zmienić podstawę systemu liczbowego w skanerze, używając metody `useRadix()`, a następnie odczytywać liczby całkowite zapisane w systemie liczbowym o tej podstawie.

```
input.useRadix(2);
System.out.print("a = ");
int a = input.nextInt();
System.out.print("b = ");
int b = input.nextInt();
```

Zadanie 11.5. Z11_5.java

Ułamek zwykły wprowadzamy z klawiatury jako łańcuch znaków (np. *23/45*), w którym licznik i mianownik (liczby całkowite) oddzielone są symbolem `/` (bez odstępów). Zakładamy, że użytkownik poprawnie wprowadzi dane. Wczytujemy token ze skanera `input` i zamieniamy na liczby całkowite `a` i `b` części łańcucha przed znakiem `/` i po nim. Aby obliczyć wartość dziesiętną ułamka, podzielimy licznik przez mianownik `a/b`. Takie dzielenie dałoby jednak wynik całkowity, więc należałoby co najmniej jedną z tych liczb skonwertować (rzutować) na typ zmiennoprzecinkowy `double` (lub

Ebookpoint.pl kopia dla: pawher@o2.pl

`float`) i uzyskać wynik zmiennoprzecinkowy, np. `(double)a/b` lub `a/(double)b`. Ponieważ dodatkowo potrzebujemy zamienić ten ułamek na procent, to mnożymy wynik przez *100*. Stosując stałą w postaci zmiennoprzecinkowej `100.0` (`double`) lub `100.0f` (`float`) i odpowiednią kolejność działań, uzyskamy właściwy rezultat (bez potrzeby jawnego rzutowania `a` lub `b` na typ zmiennoprzecinkowy): `100.0*a/b` lub `100.0f*a/b`. Mnożenie i dzielenie mają ten sam priorytet, ale działania w tym przypadku wykonywane są od strony lewej do prawej — mnożenie da wynik zmiennoprzecinkowy, więc wynik późniejszego dzielenia również będzie liczbą zmiennoprzecinkową.

```
import java.util.Scanner;
public class Z11_5 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj ułamek zwykły (np. 23/45): ");
        String u = input.next();
        int a = Integer.parseInt(u.substring(0, u.indexOf("/")));
        int b = Integer.parseInt(u.substring(u.indexOf("/")+1));
        System.out.printf("%d/%d = %.1f%%%n", a, b, 100.0*a/b);
        input.close();
    }
}
```

Zadanie 11.6. Z11_6.java

Łańcuch `s` zawiera dane tekstowe — cztery tokeny oddzielone odstępami (dwa słowa, liczbę całkowitą i liczbę dziesiętną).

```
String s = "Jan Nowak 150 25,3";
Scanner input = new Scanner(s);
```

Podając łańcuch `s` jako parametr konstruktora, otrzymamy obiekt `input` umożliwiający odczytanie danych z łańcucha znaków. Stosując metodę `next()`, odczytamy słowa (*Jan* i *Nowak*), potem odczytamy liczbę całkowitą *150* (metodą `nextInt()`) i liczbę dziesiętną (zmiennoprzecinkową) *25,3* (metodą `nextDouble()`). Mając dane, wykonamy obliczenia i sformułujemy odpowiedź. Na koniec zamykamy skaner (`input.close()`).

```
import java.util.Scanner;
public class Z11_6 {
    public static void main(String[] args) {
        String s = "Jan Nowak 150 25,3";
        Scanner input = new Scanner(s);
        String s1 = input.next();
        String s2 = input.next();
        int a = input.nextInt();
        double b = input.nextDouble();
        System.out.printf("%s %s %.2f\n", s2, s1, a*b);
        input.close();
    }
}
```

Zadanie 11.7. Z11_7.java

Rozwiązanie tego zadania jest podobne do rozwiązania zadania 11.6. Istotną różnicą jest źródło danych — tym razem dane odczytamy z pliku. Dodatkowo importujemy dwie klasy z pakietu `java.io`, klasę `IOException` (wyjątki operacji *wejścia-wyjścia*,

Ebookpoint.pl kopia dla: pawher@o2.pl

które poznasz dokładniej w dalszych rozdziałach) i klasę `File`. W nagłówku metody `main` dopisujemy informację o liście wyjątków (`throws IOException`), które mogą zostać wygenerowane podczas operacji z plikami. Obsługą tych wyjątków zajmie się *wirtualna maszyna Javy*, wywołująca metodę `main`. Sytuacja ta zostanie omówiona dokładniej na dalszych stronach.

Teraz skoncentrujmy uwagę na wykorzystaniu klasy `Scanner` do odczytywania danych z pliku. Tworzymy obiekt `f` klasy `File` skojarzony z plikiem *dane.txt* (`File f = new File("dane.txt")`) i obiekt `input` klasy `Scanner` do odczytywania danych z pliku (`Scanner input = new Scanner(f)`). Reszta kodu pozostaje bez zmian.

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
public class Z11_7 {
    public static void main(String[] args) throws IOException {
        File f = new File("dane.txt");
        Scanner input = new Scanner(f);
        String s1 = input.next();
        String s2 = input.next();
        int a = input.nextInt();
        double b = input.nextDouble();
        System.out.printf("%s %s %.2f\n", s2, s1, a*b);
        input.close();
    }
}
```

12. Operacje na tekstach — klasy StringBuffer i StringBuilder

Zadanie 12.1. Z12_1.java

To zadanie i dwa zadania następne (12.2 i 12.3) traktujemy jako pretekst do przedstawienia wybranych metod klasy `StringBuilder` (lub `StringBuffer`).

Wykresem równania $y = ax^2 + bx + c, \quad a \neq 0$ jest parabola o wierzchołku $\left(\frac{-b}{2a}, \frac{-\Delta}{4a}\right)$, gdzie $\Delta = b^2 - 4ac$. Pobieramy dane (współczynniki trójmianu) z konsoli, zakładając, że użytkownik poda poprawne liczby (`a` różne od zera), i obliczamy wyróżnik trójmianu (`delta`). Pozostałe obliczenia wykonamy podczas budowania odpowiedzi.

Tworzymy obiekt `point` klasy `StringBuilder` zawierający łańcuch znaków `"Wierzchołek paraboli ("`, czyli początek odpowiedzi.

```
StringBuilder point = new StringBuilder("Wierzchołek paraboli (");
```

Stosując metodę `append()`, dodamy na końcu łańcucha znaków (obiektu `point`) kolejno: wartość pierwszej współrzędnej wierzchołka paraboli (`-b/(2*a)`), przecinek i odstęp (`", "`), wartość drugiej współrzędnej wierzchołka paraboli (`-delta/(4*a)`) i nawias

Ebookpoint.pl kopia dla: pawher@o2.pl

zamykający parę współrzędnych punktu (")"). Zwróćmy uwagę na użycie nawiasów w mianownikach wyrażeń, wyznaczających właściwą kolejność wykonywania działań (mnożenia i dzielenia).

```
import java.util.Scanner;
public class Z12_1 {
    public static void main(String args[]) {
        System.out.println("Obliczanie współrzędnych wierzchołka
            paraboli");
        System.out.print("Podaj współczynniki trójmianu kwadratowego
            oddzielone odstępami: ");
        double a, b, c;
        /* Wczytywanie danych z konsoli */
        Scanner input = new Scanner(System.in);
        a = input.nextDouble();
        b = input.nextDouble();
        c = input.nextDouble();
        input.close();
        /* Obliczenia i budowanie odpowiedzi */
        double delta = b*b-4*a*c;
        StringBuilder point =
            new StringBuilder("Wierzchołek paraboli (");
        point.append(-b/(2*a)).append(", ").
            append(-delta/(4*a)).append(")");
        System.out.println(point);
    }
}
```

Do sformułowania odpowiedzi możemy użyć *operatora konkatenacji* (łączenia tekstów +). Kompilator w tym przypadku również wykorzysta metody klasy `StringBuilder`.

```
String s = "Wierzchołek paraboli ("+(-b/(2*a))+", "+(-delta/(4*a))+")";
System.out.println(s);
```


Uwaga

1. Zadanie możemy rozwiązać w taki sam sposób, zastępując klasę `StringBuilder` klasą `StringBuffer`. Klasy mają identyczne metody, jednak klasa `StringBuilder` jest nowsza. Po klasę `StringBuffer` sięgniemy wtedy, gdy niezbędne będzie buforowanie operacji na łańcuchu znaków.
2. Odpowiedź możemy sformułować i sformatować, stosując metodę `printf()` lub `format()` do wyświetlania wyniku:

   ```
   System.out.printf("Wierzchołek paraboli(%.2f, %.2f)", -b/(2*a),
   -delta/(4*a));
   ```

   lub metodę `format()` do budowania łańcucha:

   ```
   String s = String.format(("Wierzchołek paraboli(%.2f, %.2f)", -b/(2*a),
   -delta/(4*a));
   ```

Zadanie 12.2. Z12_2.java

Ponownie rozwiązujemy zadanie 12.1, stosując inny sposób budowania odpowiedzi. Zaczynamy od łańcucha (obiekt `point` klasy `StringBuilder`) złożonego z pary nawiasów okrągłych, przecinka i odstępu `"(, )"` — łącznie cztery znaki o indeksach `0`, `1`, `2` i `3`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Przed znakiem o indeksie 3 (między odstępem i nawiasem zamykającym) wstawiamy wartość drugiej współrzędnej wierzchołka paraboli `point.insert(3, (-delta/(4*a)))`. Konwersja zmiennoprzecinkowego typu wartości wyrażenia na łańcuch znaków wykonywana jest przez metodę `insert`. Następnie przed znakiem o indeksie 1 (między nawiasem otwierającym i przecinkiem) wstawiamy wartość pierwszej współrzędnej wierzchołka paraboli `point.insert(1, -b/(2*a))`. Obie te czynności możemy zapisać łącznie, w jednym wierszu:

```
point.insert(3, (-delta/(4*a))).insert(1, -b/(2*a));
```

Bardzo ważna jest kolejność wstawiania tych liczb do łańcucha początkowego `"(, )"`.

```
/* Obliczenia i budowanie odpowiedzi */
double delta = b*b-4*a*c;
StringBuilder point = new StringBuilder("(, )");
point.insert(3, (-delta/(4*a))).insert(1, -b/(2*a));
System.out.println("Wierzchołek paraboli: "+point);
```

Pozostała część kodu jest podobna jak w zadaniu 12.1.

Zadanie 12.3. Z12_3.java

Jeszcze raz rozwiązujemy zadanie 12.1, stosując tym razem metodę `replace()` (zastąp) z klasy `StringBuilder`. Na początek budujemy łańcuch (obiekt `point` klasy `StringBuilder`) o postaci `"(#1, #2)"`. Występujące w łańcuchu symbole `#1` i `#2` wskazują miejsca, w których umieścimy wyniki obliczeń. Metoda `replace()` ma trzy parametry, dwa indeksy określające zakres zastępowanych znaków i podstawiany w to miejsce łańcuch znaków (długość tego łańcucha nie jest ograniczona przez zakres zastępowanych znaków). Liczby przed wstawieniem musimy zamienić na łańcuchy znaków.

Najpierw znajdujemy położenie (indeks pierwszego znaku) symbolu `"#1"` w obiekcie `point` (`start = point.indexOf("#1")`), a następnie zamieniamy dwa znaki (od indeksu `start` do indeksu `start+1`) łańcuchem znaków utworzonym z wartości pierwszej współrzędnej wierzchołka paraboli (`String.valueOf(-b/(2*a))`). Podobnie zamieniamy symbol `"#2"` na odpowiednią wartość.

```
/* Obliczenia i budowanie odpowiedzi */
double delta = b*b-4*a*c;
StringBuilder point = new StringBuilder("(#1, #2)");
int start = point.indexOf("#1");
point.replace(start, start+1, String.valueOf(-b/(2*a)));
start = point.indexOf("#2");
point.replace(start, start+1, String.valueOf(-delta/(4*a)));
System.out.println("Wierzchołek paraboli: "+point);
```

W drugim wariancie rozwiązania proponujemy inny sposób zamiany liczby zmiennoprzecinkowej na łańcuch znaków (`new Double(-b/(2*a)).toString()`).

```
/* Obliczenia i budowanie odpowiedzi */
double delta = b*b-4*a*c;
StringBuilder point = new StringBuilder("(#1, #2)");
int start = point.indexOf("#1");
point.replace(start, start+1, new Double(-b/(2*a)).toString());
start = point.indexOf("#2");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
point.replace(start, start+1, new Double(-delta/(4*a)).toString());
System.out.println("Wierzchołek paraboli: "+point);
```

Kolejny wariant może wykorzystać właściwości *operatora konkatenacji* (+):

```
point.replace(start, start+1, ""+(-b/(2*a)));
```

Zadanie 12.4. Z12_4.java

Z konsoli odczytujemy liczbę całkowitą dodatnią `n` (nie sprawdzamy poprawności danych wprowadzonych przez użytkownika) i tworzymy obiekt `liczba` klasy `StringBuilder`, reprezentujący pusty łańcuch znaków.

Po wywołaniu metod `liczba.append(n).reverse()` obiekt `liczba` zawiera cyfry liczby `n`, zapisane w odwrotnej kolejności. W odpowiednim momencie zamienimy zawartość obiektu na liczbę całkowitą `Integer.parseInt(liczba.toString())`.

Drugi obiekt `wynik` klasy `StringBuilder` i metodę `append()` wykorzystamy do skonstruowania odpowiedzi.

```
import java.util.Scanner;
public class Z12_4 {
    public static void main(String args[]) {
        System.out.print("Podaj liczbę całkowitą dodatnią: ");
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        input.close();
        StringBuilder liczba = new StringBuilder();
        liczba.append(n).reverse();
        StringBuilder wynik = new StringBuilder();
        wynik.append(n).append("-").append(liczba).append(" = ");
        wynik.append(n - Integer.parseInt(liczba.toString()));
        System.out.println(wynik);
    }
}
```

Zadanie 12.5. Z12_5.java

Współrzędne wektora wprowadzamy w postaci pary liczb oddzielonych przecinkiem i zawartych w nawiasie prostokątnym, np. *[2.5, –3]*.

Z konsoli wczytujemy wiersz tekstu i usuwamy (metoda `trim()`) zbędne białe znaki z początku i końca łańcucha `String wekt = input.nextLine().trim()`. Zakładamy, że otrzymany łańcuch zawiera poprawną postać współrzędnych wektora, i tworzymy nowy obiekt, na którym wykonamy dalsze przekształcenia (`StringBuilder tmp = new StringBuilder(wekt)`).

Z łańcucha usuwamy nawiasy prostokątne, czyli pierwszy (`tmp.deleteCharAt(0)`) i ostatni znak (`tmp.deleteCharAt(tmp.length()-1)`). Następnie odszukujemy położenie przecinka oddzielającego współrzędne wektora (`int poz = tmp.indexOf(",")`) i rozdzielamy łańcuch na dwa podłańcuchy, które konwertujemy na zmiennoprzecinkowe wartości współrzędnych wektora (liczby zapisane są z kropką dziesiętną lub w postaci całkowitej):

Ebookpoint.pl kopia dla: pawher@o2.pl

```
double a = Double.parseDouble(tmp.substring(0, poz));
double b = Double.parseDouble(tmp.substring(poz+1));
```

Na koniec pozostaje obliczenie długości wektora ze wzoru $\left|\vec{u}\right| = \sqrt{a^2 + b^2}$ (`Math.sqrt(a*a+b*b)`) i sformułowanie odpowiedzi.

```
import java.util.Scanner;
public class Z12_5 {
    public static void main(String args[]) {
        System.out.print("Podaj współrzędne wektora
            (w postaci [a, b]): ");
        Scanner input = new Scanner(System.in);
        String wekt = input.nextLine().trim();
        input.close();
        StringBuilder tmp = new StringBuilder(wekt);
        /* Usuwamy pierwszy i ostatni znak, czyli nawiasy prostokątne. */
        tmp.deleteCharAt(0).deleteCharAt(tmp.length()-1);
        /* Znajdujemy pozycję przecinka między liczbami. */
        int poz = tmp.indexOf(",");
        /* Odczytujemy wartości liczb. */
        double a = Double.parseDouble(tmp.substring(0, poz));
        double b = Double.parseDouble(tmp.substring(poz+1));
        /* Obliczamy i wyświetlamy długość wektora. */
        System.out.println("Długość wektora "+wekt+" jest równa "+
            Math.sqrt(a*a+b*b));
    }
}
```

Do pobrania współrzędnych wektora z łańcucha znaków możemy wykorzystać metodę `split()` z klasy `String`. W tym celu tworzymy tablicę łańcuchów `temp[]`, konwertujemy obiekt `tmp` na łańcuch znaków i wywołujemy metodę `split()`, podając przecinek jako separator — `","`:

```
String temp[] = tmp.toString().split(",");
```

Następnie odczytujemy (z tablicy) wartości współrzędnych wektora:

```
double a = Double.parseDouble(temp[0]);
double b = Double.parseDouble(temp[1]);
```

Pozostała część kodu pozostaje bez zmian.

Stosując metody `substring()` i `split()`, możemy zrezygnować z użycia obiektu `tmp` klasy `StringBuilder`:

```
String temp[] = wekt.substring(1, wekt.length()-1).split(",");
```

Oczywiście dane muszą być wprowadzane zgodnie z przyjętą umową.

Zadanie 12.6. Z12_6.java

Zadanie to rozwiązujemy podobnie jak zadanie 12.5. Istotna różnica polega na tym, że mamy trzy współrzędne oddzielone dwoma przecinkami. Położenie tych przecinków (pierwszego i ostatniego) ustalimy, używając metod `indexOf()` i `lastIndexOf()`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;

public class Z12_6 {
    public static void main(String args[]) {
        System.out.print("Podaj współrzędne wektora
            (w postaci [a, b, c]): ");
        Scanner input = new Scanner(System.in);
        String wekt = input.nextLine().trim();
        input.close();
        StringBuilder tmp = new StringBuilder(wekt);
        /* Usuwamy pierwszy i ostatni znak, czyli nawiasy prostokątne. */
        tmp.deleteCharAt(0).deleteCharAt(tmp.length()-1);
        /* Znajdujemy pozycje przecinków między liczbami. */
        int poz1 = tmp.indexOf(",");
        int poz2 = tmp.lastIndexOf(",");
        /* Odczytujemy wartości liczb. */
        double a = Double.parseDouble(tmp.substring(0, poz1));
        double b = Double.parseDouble(tmp.substring(poz1+1, poz2));
        double c = Double.parseDouble(tmp.substring(poz2+1));
        /* Obliczamy i wyświetlamy długość wektora. */
        System.out.println("Długość wektora "+wekt+" jest równa "+
            Math.sqrt(a*a+b*b+c*c));
    }
}
```

Podobnie jak w rozwiązaniu zadania 12.5, do wyznaczenia współrzędnych wektora możemy zastosować metodę `split()` z klasy `String`:

```
/* Pomijamy pierwszy i ostatni znak, czyli nawiasy prostokątne, i dzielimy łańcuch znaków
 na części rozdzielone przecinkiem. */
String temp[] = wekt.substring(1, wekt.length()-1).split(",");
/* Odczytujemy wartości liczb - współrzędnych wektora. */
double a = Double.parseDouble(temp[0]);
double b = Double.parseDouble(temp[1]);
double c = Double.parseDouble(temp[2]);
```

13. Instrukcje warunkowe i instrukcja selekcji

Zadanie 13.1. Z13_1.java

Wprowadzone imię jest przechowywane w obiekcie `str` klasy `String`. Odpowiedź skonstruujemy jako obiekt `name` klasy `StringBuilder` — `StringBuilder name = new StringBuilder(str);`. W zmiennej `z` przechowujemy ostatni znak imienia — `char z = name.charAt(name.length()-1);`. Do imienia (`name`) dodajemy słowo `" jest "` wraz z odstępami przed słowem i za słowem — `name.append(" jest ");`. Na koniec uzupełniamy odpowiedź słowem określającym płeć osoby, w zależności od tego, jaką wartość ma wyrażenie logiczne `z == 'a'`. Jeśli wyrażenie ma wartość `true` (ostatnią literą imienia jest `a`), to do odpowiedzi dodajemy tekst `"kobietą."`, w przeciwnym razie dodajemy tekst `"mężczyzną."`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;

public class Z13_1 {
    public static void main(String args[]) {
        System.out.print("Podaj imię: ");
        Scanner input = new Scanner(System.in);
        String str = input.next();
        input.close();
        StringBuilder name = new StringBuilder(str);
        char z = name.charAt(name.length()-1);
        name.append(" jest ");
        if (z == 'a')
            name.append("kobietą.");
        else
            name.append("mężczyzną.");
        System.out.println(name);
    }
}
```

Zadanie 13.2. Z13_2.java

Jeżeli wczytana wartość zmiennej `a` (długość boku kwadratu) jest dodatnia, to wykonujemy obliczenia i wyświetlamy wynik. W kolejnej instrukcji warunkowej badamy, czy podana długość boku jest mniejsza lub równa 0 — jeśli tak jest, to wyświetlamy komunikat o błędnych danych.

```
import java.util.Scanner;

public class Z13_2 {
    public static void main(String args[]) {
        System.out.println("Obliczanie pola powierzchni i obwodu kwadratu");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj długość boku, a = ");
        double a = input.nextDouble();
        input.close();
        if (a > 0) {
            double pole = a*a;
            double obwod = 4*a;
            System.out.println("Pole powierzchni P = "+pole);
            System.out.println("Obwód kwadratu L = "+obwod);
        }
        if (a <= 0)
            System.out.println("Błąd! Długość boku ma być liczbą dodatnią.");
    }
}
```

Ponieważ warunki `a > 0` i `a <= 0` nawzajem wykluczają się, to bardzo często używamy w takiej sytuacji konstrukcji `if-else`:

```
if (a > 0) {
    double pole = a*a;
    double obwod = 4*a;
    System.out.println("Pole powierzchni P = "+pole);
    System.out.println("Obwód kwadratu L = "+obwod);
} else
    System.out.println("Błąd! Długość boku ma być liczbą dodatnią.");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.3. Z13_3.java

Po wprowadzeniu danych tworzymy obiekt `wynik` klasy `StringBuilder`. Stanowiący odpowiedź łańcuch znaków składa się z nawiasów < i > dla przedziału domkniętego lub { i } w przypadku zbioru jednoelementowego (przypadek `a` jest równe `b`). Wartości zmiennych `a` i `b` w przypadku przedziału powinny być wprowadzone do wyniku w porządku rosnącym. Mamy zatem trzy przypadki:

Przypadek	Budowanie odpowiedzi
`a < b`	`wynik.append("<").append(a).append(", ").append(b).append(">");`
`b < a`	`wynik.append("<").append(b).append(", ").append(a).append(">");.`
`a == b`	`wynik.append("{").append(a).append("}");`

Przedstawiona sytuacja wymaga zbudowania trzech instrukcji warunkowych lub zastosowania tzw. zagnieżdżenia instrukcji warunkowej.

```
import java.util.Scanner;

public class Z13_3 {
    public static void main(String args[]) {
        System.out.println("Podaj dwie liczby a i b:");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        System.out.print("b = ");
        double b = input.nextDouble();
        input.close();
        StringBuilder wynik = new StringBuilder();
        if (a < b)
            wynik.append("<").append(a).append(", ").append(b).append(">");
        else if (b < a)
            wynik.append("<").append(b).append(", ").append(a).append(">");
        else
            wynik.append("{").append(a).append("}");
        wynik.insert(0, "Zbiór wszystkich liczb zawartych pomiędzy a i b, X = ");
        System.out.println(wynik);
    }
}
```

Zadanie 13.4. Z13_4.java

Po wprowadzeniu wartości współczynnika `a` trójmianu ax^2+bx+c sprawdzamy podaną wartość. Jeśli `a` jest zerem, to wyświetlamy komunikat skierowany do standardowego strumienia błędów `System.err.println(...)` i zamykamy aplikację `System.exit(0)`. Gdy `a` jest różne od zera, wczytujemy pozostałe współczynniki trójmianu (`b` i `c`) i obliczamy $q=\frac{-\Delta}{4a}=\frac{-\left(b^2-4ac\right)}{4a}=\frac{4ac-b^2}{4a}$. Zbiór wartości funkcji zależy od znaku współczynnika `a` i wartości `q`:

Ebookpoint.pl kopia dla: pawher@o2.pl

Znak a	Zbiór wartości	Budowanie odpowiedzi
a > 0	$\langle q, +\infty)$	wynik.append("<, +oo)").insert(1, q);
a < 0	$(-\infty, q\rangle$	wynik.append("(-oo, >").insert(6, q);

```
import java.util.Scanner;
public class Z13_4 {
    public static void main(String args[]) {
        System.out.println("Podaj współczynniki funkcji kwadratowej:");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        if (a == 0) {
            System.err.println("a = 0, to nie jest funkcja kwadratowa.");
            System.exit(0);
        }
        System.out.print("b = ");
        double b = input.nextDouble();
        System.out.print("c = ");
        double c = input.nextDouble();
        input.close();
        double q = (4*a*c-b*b)/(4*a);
        StringBuilder wynik = new StringBuilder();
        if (a > 0)
            wynik.append("<, +oo)").insert(1, q);
        else
            wynik.append("(-oo, >").insert(6, q);
        System.out.println("Zbiór wartości: "+wynik);
    }
}
```

Zadanie 13.5. Z13_5.java

Wczytywanie danych (współczynników trójmianu) zorganizujemy tak, jak w rozwiązaniu zadania 13.4. Następnie obliczymy współrzędne `p` i `q` wierzchołka paraboli będącej wykresem tej funkcji kwadratowej.

```
double p = -b/(2*a);
double q = (4*a*c-b*b)/(4*a);
```

W budowanej odpowiedzi umieścimy symbole `#1` i `#2`. W miejsce tych symboli podstawimy później odpowiednie słowa `"rosnąca"` i `"malejąca"`, w zależności od znaku współczynnika `a`.

```
StringBuilder wynik = new StringBuilder();
wynik.append("Funkcja jest #1 w przedziale (-oo, ").append(p);
wynik.append("> i #2 w przedziale <").append(p).append(", +oo).");
int p1 = wynik.indexOf("#1");
int p2 = wynik.indexOf("#2");
if (a > 0)
    wynik.replace(p2, p2+2, "rosnąca").replace(p1, p1+2, "malejąca");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
else
    wynik.replace(p2, p2+2, "malejąca").replace(p1, p1+2, "rosnąca");
System.out.println(wynik);
```

Usuwamy wszystkie znaki z obiektu `wynik` i budujemy drugą odpowiedź.

```
wynik.delete(0, wynik.length());
wynik.append("Funkcja osiąga wartość ");
if (a > 0)
    wynik.append("minimalną");
else
    wynik.append("maksymalną");
wynik.append(" y = ").append(q).append(" w punkcie x = ").append(p).append(".");
System.out.println(wynik);
```

Zadanie 13.6. Z13_6.java

Równanie liniowe o postaci $ax + b = 0$ jest równoważne równaniu $ax = -b$. Jeżeli $a \neq 0$, to rozwiązaniem równania jest liczba $x = \frac{-b}{a}$, w przeciwnym razie otrzymujemy równanie $0x = -b$. Dla $b \neq 0$ równanie jest sprzeczne (*0x=1* — żadna liczba nie spełnia równania), a dla *b=0* równanie jest spełnione przez dowolną liczbę *x* (*0x=0* — równanie tożsamościowe). Zależności te przedstawimy przy użyciu instrukcji warunkowej:

```
if (a != 0) {
    double x = -b/a;
    System.out.println("x = "+x);
} else if (b != 0)
    System.out.println("Równanie sprzeczne (0x = 1).");
else
    System.out.println("Równanie tożsamościowe (0x = 0).");
```

Wczytywanie współczynników równania z konsoli zrealizujemy, stosując obiekty i metody klasy `Scanner`.

```
import java.util.Scanner;
public class Z13_6 {
    public static void main(String args[]) {
        System.out.println("Podaj współczynniki równania ax+b = 0");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        System.out.print("b = ");
        double b = input.nextDouble();
        input.close();
        if (a != 0) {
            double x = -b/a;
            System.out.println("x = "+x);
        } else if (b != 0)
            System.out.println("Równanie sprzeczne (0x = 1).");
        else
            System.out.println("Równanie tożsamościowe (0x = 0).");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.7. Z13_7.java

Najpierw porównujemy pierwszą i drugą liczbę (a i b), a następnie mniejszą z nich porównujemy z liczbą c. Te dwa porównania wystarczą, aby wskazać najmniejszą z trzech liczb.

```
import java.util.Scanner;

public class Z13_7 {
    public static void main(String args[]) {
        System.out.println("Wybór najmniejszej z trzech liczb.");
        System.out.println("Podaj liczby:");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        System.out.print("b = ");
        double b = input.nextDouble();
        System.out.print("c = ");
        double c = input.nextDouble();
        input.close();
        System.out.print("Najmniejszą liczbą jest ");
        if (a < b)
            if (a < c)
                System.out.println(a);
            else
                System.out.println(c);
        else
            if (b < c)
                System.out.println(b);
            else
                System.out.println(c);
    }
}
```

Użyte instrukcje warunkowe możemy zastąpić wyrażeniem warunkowym. Wyrażenie `(a < b)?a:b` zwraca mniejszą spośród liczb a i b. Ta wartość jest porównywana w taki sam sposób z liczbą c: `(((a < b)?a:b) < c)?((a < b)?a:b):c)`.

```
System.out.print("Najmniejszą liczbą jest: ");
System.out.println((((a < b)?a:b) < c)?((a < b)?a:b):c);
```

Zamieniając znak < na znak > w powyższym kodzie, otrzymamy program wyznaczający największą z trzech liczb.

Zadanie 13.8. Z13_8.java

Dane możemy wprowadzić w sposób przedstawiony w rozwiązaniu zadania 13.7. Ze zbioru trzech liczb parę liczb możemy wyznaczyć na trzy sposoby, czyli co najwyżej po trzech porównaniach poznamy kolejność (uporządkowanie) tych liczb.

```
System.out.print("Liczby uporządkowane w kolejności niemalejącej: ");
    if (a <= b)
        if (a <= c)
            if (b <= c)
                System.out.printf("%f, %f, %f.\n", a, b, c);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            else
                System.out.printf("%f, %f, %f.\n", a, c, b);
        else
            System.out.printf("%f, %f, %f.\n", c, a, b);
    else
        if (b <= c)
            if (a <= c)
                System.out.printf("%f, %f, %f.\n", b, a, c);
            else
                System.out.printf("%f, %f, %f.\n", b, c, a);
        else
            System.out.printf("%f, %f, %f.\n", c, b, a);
```

Zastosowaną w rozwiązaniu metodę wyświetlania wyników `System.out.printf("%f, %f, %f.\n", a, b, c);` możemy również zrealizować, wykorzystując metodę `println()` — `System.out.println(a+", "+b+", "+c+".");`.

Zamieniając znak <= na znak => w powyższym kodzie, otrzymamy program porządkujący trzy liczby w kolejności nierosnącej.

Zadanie 13.9. Z13_9.java

Mamy dziesięć liczb jednocyfrowych (*0*, *1*, …, *9*) i każdej z nich należy przyporządkować jedno słowo (*zero*, *jeden*, …, *dziewięć*). Możemy to zrobić metodą „kopiuj, wklej i popraw", tworząc ciąg dziesięciu instrukcji warunkowych:

```
String wynik = "";
if (n == 0) wynik = "zero";
if (n == 1) wynik = "jeden";
// itd.
if (n == 9) wynik = "dziewięć";
System.out.println("Słownie: "+wynik);
```

Niezależnie od wartości liczby `n` porównanie wykonujemy dziesięć razy. Efektywniejsze będzie zastosowanie instrukcji warunkowych zagnieżdżonych:

```
String wynik = "";
if (n == 0) wynik = "zero";
else if (n == 1) wynik = "jeden";
// itd.
else if (n == 8) wynik = "dziewięć";
else wynik = "dziewięć"; // pozostała ostatnia możliwość
System.out.println("Słownie: "+wynik);
```

W tym przypadku liczba porównań będzie zależała od wartości `n` i zmienia się w zakresie od *1* do *9* (zakładamy, że otrzymaliśmy do zamiany liczbę jednocyfrową i ostatnie porównanie `n == 9` jest zbędne).

Do wykonywania takich porównań możemy użyć instrukcji wyboru `switch()`:

```
import java.util.Scanner;

public class Z13_9 {
    public static void main(String args[]) {
        System.out.println("Słowny zapis liczby jednocyfrowej");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj liczbę naturalną jednocyfrową, n = ");
        int n = input.nextInt();
        StringBuilder wynik = new StringBuilder("Słownie: ");
        if (n >=0 && n < 10) {
            switch (n) {
                case 0: wynik.append("zero"); break;
                case 1: wynik.append("jeden"); break;
                case 2: wynik.append("dwa"); break;
                case 3: wynik.append("trzy"); break;
                case 4: wynik.append("cztery"); break;
                case 5: wynik.append("pięć"); break;
                case 6: wynik.append("sześć"); break;
                case 7: wynik.append("siedem"); break;
                case 8: wynik.append("osiem"); break;
                case 9: wynik.append("dziewięć"); break;
            }
            System.out.println(wynik.append("."));
        } else
            System.out.println("Błędna wartość liczby!");
    }
}
```

Zwróćmy uwagę na użycie instrukcji warunkowej `if (n >=0 && n < 10) {...}` w celu sprawdzenia, czy liczba `n` jest jednocyfrowa (większa lub równa *0* i mniejsza od *10*).

Możemy zamiast instrukcji wyboru zastosować tablicę łańcuchów (możemy zdefiniować tę tablicę w kodzie klasy jako prywatną statyczną stałą `private static final`):

```
String[] str = {"zero", "jeden", "dwa", "trzy", "cztery", "pięć", "sześć",
"siedem", "osiem", "dziewięć"};
```

i jako wynik wybierać odpowiednią wartość z tej tablicy `str[n]` (dla `n = 0`, `1`, …, `9`).

```
StringBuilder wynik = new StringBuilder("Słownie: ");
if (n >=0 && n < 10)
    System.out.println(wynik.append(str[n]).append("."));
else
    System.out.println("Błędna wartość liczby!");
```

Można pominąć obiekt `wynik` klasy `StringBuilder` i bezpośrednio sformułować odpowiedź:

```
if (n >=0 && n < 10)
    System.out.println("Słownie: "+str[n]+".");
else
    System.out.println("Błędna wartość liczby!");
```

Nie zalecamy jednak tej oszczędności ze względu na kolejne zadania wykorzystujące omówione fragmenty kodu.

Zadanie 13.10. Z13_10.java

Przeanalizujmy zapis słowny liczby dwucyfrowej (tj. jednej z dziewięćdziesięciu liczb od *10* do *99*). Możemy mieć do zapisania słownie pełne dziesiątki (*10*, *20*, …, *90*), pełne dziesiątki od *20* w górę i jedności (zob. rozwiązanie zadania 13.9) oraz liczby

Ebookpoint.pl kopia dla: pawher@o2.pl

z drugiej dziesiątki (*11*, *12*, …, *19*), wymagające odrębnego potraktowania. Do tej ostatniej grupy dołączymy liczbę *10*.

Z liczby dwucyfrowej `n` możemy obliczyć jej cyfrę jedności `j = n%10` (reszta z dzielenia przez *10*) oraz cyfrę dziesiątek `d = n/10` (wynik dzielenia całkowitego `n` przez *10*, `d` równe zero, świadczy o tym, że liczba `n` jest jednocyfrowa).

Stosując instrukcje warunkowe, wyodrębnimy wszystkie istotne przypadki. Zaczniemy od wyeliminowania liczb o większej liczbie cyfr i liczb ujemnych:

```
if (n >= 100 || n < 0) {
    /* Błąd. To nie jest liczba naturalna dwucyfrowa. */
} else if (n >= 20) {
    /* Liczba dwucyfrowa większa lub równa 20 */
} else if(n >= 10) {
    /* 10 lub liczba z drugiej dziesiątki: 11, 12, ..., 19 */
} else {
    /* Pozostały liczby jednocyfrowe. */
}
```

Wyrażone słowami wartości liczb możemy uzyskać, stosując instrukcje selekcji lub tablice zawierające poszczególne nazwy (zob. rozwiązanie zadania 13.9).

```
import java.util.Scanner;
public class Z13_10 {
    public static void main(String args[]) {
        System.out.println("Słowny zapis liczby naturalnej mniejszej od 100");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj liczbę naturalną mniejszą od 100, n = ");
        int n = input.nextInt();
        StringBuilder wynik = new StringBuilder("Słownie: ");
        if (n >= 100 || n < 0) {
            /* Błąd. To nie jest liczba naturalna dwucyfrowa. */
            System.out.println("Błędna wartość liczby!");
        } else if (n >= 20) {
            /* Liczba dwucyfrowa większa lub równa 20 */
            switch (n/10) { // Cyfra dziesiątek
                case 2: wynik.append("dwadzieścia"); break;
                case 3: wynik.append("trzydzieści"); break;
                case 4: wynik.append("czterdzieści"); break;
                case 5: wynik.append("pięćdziesiąt"); break;
                case 6: wynik.append("sześćdziesiąt"); break;
                case 7: wynik.append("siedemdziesiąt"); break;
                case 8: wynik.append("osiemdziesiąt"); break;
                case 9: wynik.append("dziewięćdziesiąt"); break;
            }
            int d = n%10; // Cyfra jedności
            if (d > 0) {
                wynik.append(" ");
                switch (d) {
                    case 1: wynik.append("jeden"); break;
                    case 2: wynik.append("dwa"); break;
                    case 3: wynik.append("trzy"); break;
                    case 4: wynik.append("cztery"); break;
                    case 5: wynik.append("pięć"); break;
                    case 6: wynik.append("sześć"); break;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                    case 7: wynik.append("siedem"); break;
                    case 8: wynik.append("osiem"); break;
                    case 9: wynik.append("dziewięć"); break;
                }
            }
        } else if(n >= 10) {
            /* 10 lub liczba z drugiej dziesiątki: 11, 12, ..., 19 */
            switch (n) {
                case 10: wynik.append("dziesięć"); break;
                case 11: wynik.append("jedenaście"); break;
                case 12: wynik.append("dwanaście"); break;
                case 13: wynik.append("trzynaście"); break;
                case 14: wynik.append("czternaście"); break;
                case 15: wynik.append("piętnaście"); break;
                case 16: wynik.append("szesnaście"); break;
                case 17: wynik.append("siedemnaście"); break;
                case 18: wynik.append("osiemnaście"); break;
                case 19: wynik.append("dziewiętnaście"); break;
            }
        } else {
            /* Pozostały liczby jednocyfrowe. */
            switch (n) {
                case 0: wynik.append("zero"); break;
                case 1: wynik.append("jeden"); break;
                case 2: wynik.append("dwa"); break;
                case 3: wynik.append("trzy"); break;
                case 4: wynik.append("cztery"); break;
                case 5: wynik.append("pięć"); break;
                case 6: wynik.append("sześć"); break;
                case 7: wynik.append("siedem"); break;
                case 8: wynik.append("osiem"); break;
                case 9: wynik.append("dziewięć"); break;
            }
        }
        wynik.append(".");
        System.out.println(wynik);
    }
}
```

Zdecydowanie krótszy kod uzyskamy, deklarując dwie tablice zawierające potrzebne liczebniki.

```
import java.util.Scanner;
public class Z13_10a {
    private static final String[] str1 = {"zero", "jeden", "dwa", "trzy",
    "cztery", "pięć", "sześć", "siedem", "osiem", "dziewięć", "dziesięć",
    "jedenaście", "dwanaście", "trzynaście", "czternaście", "piętnaście",
    "szesnaście", "siedemnaście", "osiemnaście", "dziewiętnaście"};

    private static final String[] str2 = {"", "", "dwadzieścia",
    "trzydzieści", "czterdzieści", "pięćdziesiąt", "sześćdziesiąt",
    "siedemdziesiąt", "osiemdziesiąt", "dziewięćdziesiąt"};

    public static void main(String args[]) {
        System.out.println("Słowny zapis liczby naturalnej mniejszej od 100");
        Scanner input = new Scanner(System.in);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.print("Podaj liczbę naturalną mniejszą od 100, n = ");
        int n = input.nextInt();
        StringBuilder wynik = new StringBuilder("Słownie: ");
        if (n >= 100 || n < 0) { /* Błędne dane */
            System.out.println("Błędna wartość liczby!");
        } else if (n >= 20) { /* Liczba większa lub równa 20 */
            wynik.append(str2[n/10]);
            int d = n%10; /* Cyfra jedności */
            if (d > 0)
                wynik.append(" ").append(str1[d]);
        } else if(n >= 10) /* 10, 11, 12, ..., 19 */
            wynik.append(str1[n]);
        else /* Liczby jednocyfrowe */
            wynik.append(str1[n]);
        wynik.append(".");
        System.out.println(wynik);
    }
}
```

W pierwszej tablicy zawarto liczby od *0* do *19*, a w drugiej pełne dziesiątki od *20* do *90*. Aby uniknąć obliczeń indeksów, dwie początkowe wartości (w drugiej tablicy) są łańcuchami pustymi.

Zadanie 13.11. Z13_11.java

Wystarczy uzupełnić rozwiązanie zadania 13.10 o kod odpowiedzialny za prawidłowe odczytywanie (słownie) cyfry setek w liczbie trzycyfrowej i dołączyć do tego słowny zapis dwucyfrowej reszty.

```
StringBuilder wynik = new StringBuilder("Słownie: ");
if (n >= 1000 || n < 0) {
    /* Błąd. To nie jest liczba naturalna mniejsza od 1000. */
    System.out.println("Błędna wartość liczby!");
} else if (n == 0) {
    /* Zero */
} else {
    int s = n/100; // Cyfra setek
    if (s != 0)
        switch (s) {
            case 1: wynik.append("sto"); break;
            case 2: wynik.append("dwieście"); break;
            case 3: wynik.append("trzysta"); break;
            case 4: wynik.append("czterysta"); break;
            case 5: wynik.append("pięćset"); break;
            case 6: wynik.append("sześćset"); break;
            case 7: wynik.append("siedemset"); break;
            case 8: wynik.append("osiemset"); break;
            case 9: wynik.append("dziewięćset"); break;
        }
    n = n%100;
    if (n != 0) {
        /* Dalej zajmiemy się dwucyfrową resztą. */
        int d = n/10; // Cyfra dziesiątek
        int j = n%10; // Cyfra jedności
        if (d == 1) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            /* Liczba od 10 do 19 */
        } else {
            if (d != 0) {
                wynik.append(" ");
                /* Dopisujemy pełne dziesiątki: 20, 30, ..., 90. */
            }
            if (j != 0) {
                wynik.append(" ");
                /* Dopisujemy jedności: 1, 2, ..., 9. */
            }
        }
    }
}
wynik.append(".");
System.out.println(wynik);
```

W wariancie rozwiązania z zastosowaniem tablic (zob. rozwiązanie zadania 13.10) dodajemy tablicę `str3` zawierającą liczebniki określające pełne setki:

```
private static final String[] str3 = {"", "sto", "dwieście", "trzysta",
    "czterysta", "pięćset", "sześćset", "siedemset", "osiemset",
    "dziewięćset"};
```

i wykorzystujemy liczebniki zapisane w tablicach do słownego zapisywania liczby:

```
StringBuilder wynik = new StringBuilder("Słownie: ");
if (n >= 1000 || n < 0) /* Błąd - niewłaściwa liczba */
    System.out.println("Błędna wartość liczby!");
else if (n == 0) /* Zero */
    wynik.append(str1[0]);
else {
    int s = n/100; // Cyfra setek
    if (s != 0)
        wynik.append(str3[s]);
    n = n%100;
    if (n != 0) {
        /* Dalej zajmiemy się dwucyfrową resztą. */
        int d = n/10; // Cyfra dziesiątek
        int j = n%10; // Cyfra jedności
        if (d == 1) /* Dopisujemy liczby od 10 do 19. */
            wynik.append(" ").append(str2[n]);
        else {
            if (d != 0) /* Dopisujemy pełne dziesiątki: 20, 30,..., 90. */
                wynik.append(" ").append(str2[d]);
            if (j != 0) /* Dopisujemy jedności: 1, 2, ..., 9. */
                wynik.append(" ").append(str1[j]);
        }
    }
}
wynik.append(".");
System.out.println(wynik);
```

Zwróćmy uwagę, że odrębne potraktowanie wartości *0* (w porównaniu z rozwiązaniem zadania 13.10) pozwoliło na uproszczenie algorytmu słownego zapisywania liczb mniejszych od 100.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 13.12. Z13_12.java

Sprawdzanie daty (łańcucha znaków) wykonujemy w kilku etapach. Efekt kontroli przechowujemy w zmiennej logicznej `isDate` typu `boolean` (wartość `true` oznacza poprawną datę, `false` — błędną). Do kolejnego etapu testu przechodzimy pod warunkiem, że poprzedni etap zakończył się sukcesem. Kolejność sprawdzania przedstawia się następująco:

- długość łańcucha znaków (10);
- obecność cyfr dziesiętnych i kropek na odpowiednich pozycjach;
- sprawdzenie zakresu dla liczby wyrażającej rok oraz numer miesiąca i dnia.

Na koniec zależnie od wartości zmiennej `isDate` wyświetlamy odpowiedni komunikat.

```
import java.util.Scanner;
public class Z13_12 {
    public static void main(String args[]) {
        System.out.println("Zmiana formatu daty");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj datę w formacie dd.mm.rrrr: ");
        String str = input.next();
        /* Sprawdzamy, czy podany łańcuch znaków ma właściwą długość
           (10 znaków). */
        boolean isDate = str.length() == 10;
        /* Jeśli łańcuch ma właściwą długość, to sprawdzamy,
           czy składa się z właściwych znaków. Znaki o indeksie 2 i 5
           powinny być kropkami, a pozostałe znaki cyframi. */
        if (isDate)
            isDate = Character.isDigit(str.charAt(0))
                  && Character.isDigit(str.charAt(1))
                  && str.charAt(2) == '.'
                  && Character.isDigit(str.charAt(3))
                  && Character.isDigit(str.charAt(4))
                  && str.charAt(5) == '.'
                  && Character.isDigit(str.charAt(6))
                  && Character.isDigit(str.charAt(7))
                  && Character.isDigit(str.charAt(8))
                  && Character.isDigit(str.charAt(9));
        /* Jeśli łańcuch znaków jest poprawnie zbudowany, to wyznaczamy
           wartości zmiennych dz (dzień), mc (miesiąc) i rok. Następnie
           sprawdzamy poprawność tych danych. */
        if (isDate) {
            int dz = Integer.parseInt(str.substring(0, 2));
            int mc = Integer.parseInt(str.substring(3, 5));
            int rok = Integer.parseInt(str.substring(6, 10));
            /* Sprawdzamy, czy rok należy do żądanego stulecia (wieku),
               czy numer miesiąca należy do zakresu od 1 do 12 oraz czy
               numer dnia nie przekracza 31. */
            isDate = rok >= 2000 && rok < 2100
                  && isDate && mc >= 1 && mc <= 12
                  && dz > 0 && dz <= 31;
            /* Dla kwietnia, czerwca, września i października sprawdzamy,
               czy numer dnia nie przekracza 30. */
            if (mc == 4 || mc == 6 || mc == 9 || mc == 11)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                isDate = isDate && dz <= 30;
            /* Dla lutego w roku przestępnym sprawdzamy, czy numer dnia
               nie przekracza 29. */
            if (mc == 2 && rok%4 == 0)
                isDate = isDate && dz <= 29;
            /* Dla lutego w roku zwykłym sprawdzamy, czy numer dnia nie
               przekracza 28. */
            if (mc == 2 && rok%4 != 0)
                isDate = isDate && dz <= 28;
        }
        /* W zależności od wartości zmiennej logicznej isDate wyświetlamy
           stosowną odpowiedź. */
        if (isDate)
            System.out.println(str+" - poprawna data");
        else
            System.out.println(str+" - błędna data");
    }
}
```

Kod możemy skrócić, rezygnując ze sprawdzania, czy w zapisie daty zastosowano cyfry. Podczas parsowania ciągów błędny łańcuch `dd`, `mm` lub `rrrr` spowoduje przerwanie pracy programu — zostanie zgłoszony odpowiedni wyjątek. Do sprawdzania poprawności liczb możemy użyć instrukcji wyboru. Oto zmieniony fragment kodu:

```
/* Sprawdzamy, czy podany łańcuch znaków ma właściwą długość (10 znaków)
  oraz czy znaki o indeksie 2 i 5 są kropkami. */
boolean isDate = str.length() == 10 && str.charAt(2) == '.'
    && str.charAt(5) == '.';
/* Jeśli łańcuch znaków ma postać dd.mm.rrrr, to próbujemy zamienić ciągi
  znaków dd, mm i rrrr na liczby całkowite, a następnie sprawdzimy
  wartości tych liczb. */
if (isDate) {
    int dz = Integer.parseInt(str.substring(0, 2));
    int mc = Integer.parseInt(str.substring(3, 5));
    int rok = Integer.parseInt(str.substring(6, 10));
    /* Sprawdzamy, czy rok należy do żądanego stulecia (wieku), czy numer
       miesiąca należy do zakresu od 1 do 12 oraz czy numer dnia jest
       większy od 0. */
    isDate = rok >= 2000 && rok < 2100 && mc >= 1 && mc <= 12 && dz > 0;
    /* Sprawdzamy, czy numer dnia nie przekracza liczby dni w miesiącu. */
    switch (mc) {
        case 2:                                 // luty
            if (rok%4 == 0)
                isDate = isDate && dz <= 29; // w roku przestępnym
            else
                isDate = isDate && dz <= 28; // w pozostałych latach
            break;
        case 4: case 6: case 9: case 11:        // kwiecień, czerwiec
            isDate = isDate && dz <= 30;        // wrzesień, listopad
            break;
        default:
            isDate = isDate && dz <= 31;        // pozostałe miesiące
            break;
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Jeśli jednak nie chcemy, aby błędnie podana data (np. 12.o6.2013 — litera o zamiast cyfry 0) powodowała przerwanie programu, to musimy sprawdzać poprawność znaków lub przechwycić i obsłużyć wyjątek.

```
if (isDate) {
    try {
        int dz = Integer.parseInt(str.substring(0, 2));
        int mc = Integer.parseInt(str.substring(3, 5));
        int rok = Integer.parseInt(str.substring(6, 10));
        /* Sprawdzenie pozostawiamy bez zmian; będzie wykonywane tylko
           wtedy, gdy powyższe instrukcje wykonają się poprawnie. */
    } catch (NumberFormatException e) {
         isDate = false;
    }
}
```

Zadanie 13.13. Z13_13.java

W wejściowym łańcuchu znaków o postaci dd.mm.rrrr (zakładamy jego poprawność) musimy zastąpić kropki pauzami (dd-mm-rrrr) oraz przestawić (zmienić miejscami) podciągi dd i rrrr (mają różne długości!). Efektem będzie ciąg znaków o postaci rrrr-mm-dd.

```
import java.util.Scanner;

public class Z13_13 {
    public static void main(String args[]) {
        System.out.println("Zmiana formatu daty");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj datę w formacie dd.mm.rrrr: ");
        String str = input.next();
        /* Zakładamy poprawność formatu i wartości wprowadzonej daty. */
        StringBuilder data = new StringBuilder(str);
        data.replace(2, 3, "-").replace(5, 6, "-");
        data.append(data.substring(0,2)).delete(0, 2);
        data.insert(0, data.substring(4, 8)).delete(8, 12);
        System.out.println(data);
     }
}
```

Prześledźmy działanie programu na przykładzie.

Wykonywana operacja	Obiekt data 111111 (index) 0123456789012345
StringBuilder data = new StringBuilder(str)	21.01.2012
data.replace(2, 3, "-")	21-01.2012
data.replace(5, 6, "-")	21-01-2012
data.append(data.substring(0,2))	21-01-201221
data.delete(0, 2)	-01-201221
data.insert(0, data.substring(4, 8))	2012-01-201221
data.delete(8, 12)	2012-01-21

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Metoda `substring()` zwraca odpowiedni podłańcuch znaków z obiektu `data`, ale nie modyfikuje wartości obiektu. Pozostałe zastosowane metody (`append()`, `delete()`, `insert()` i `replace()`) zmieniają wartość obiektu `data`.

W metodach `replace()` i `delete()` pierwszy parametr określa indeks znaku, od którego zaczynamy operację, a drugi parametr jest indeksem znaku, przed którym kończymy wykonywanie operacji, np. `replace(2, 3, "-")` zamienia wyłącznie znak o indeksie `2` na wskazany łańcuch, a `delete(8, 12)` usuwa znaki o indeksach `8`, `9`, `10` i `11`. Metoda `insert()` wstawia łańcuch znaków przed znakiem o wskazanym indeksie.

Zadanie 13.14. Z13_14.java

a) Zakładamy, że użytkownik wprowadził poprawną datę w postaci ciągu znaków `str` (`dd.mm.rrrr`). Wyznaczamy numer miesiąca:

```
int mc = Integer.parseInt(str.substring(3, 5));
```

i tworzymy obiekt `data` klasy `StringBuilder` zawierający wprowadzoną datę.

W obiekcie `data` zamieniamy *kropki* (znaki o indeksie *2* i *5*, oddzielające elementy daty) na *odstępy* oraz zastępujemy znaki o indeksach *3* i *4* (`rr`) na słowną nazwę miesiąca. Należy pamiętać, że w zapisie słownym daty miesiąc podajemy w dopełniaczu.

```
import java.util.Scanner;
public class Z13_14 {
    public static void main(String args[]) {
        System.out.println("Zmiana formatu daty");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj datę w formacie dd.mm.rrrr: ");
        String str = input.next();
        /* Zakładamy poprawność formatu i wartości wprowadzonej daty. */
        int mc = Integer.parseInt(str.substring(3, 5));
        StringBuilder data = new StringBuilder(str);
        data.replace(2, 3, " ").replace(5, 6, " ");
        switch (mc) {
            case  1: data.replace(3, 5, "stycznia"); break;
            case  2: data.replace(3, 5, "lutego"); break;
            case  3: data.replace(3, 5, "marca"); break;
            case  4: data.replace(3, 5, "kwietnia"); break;
            case  5: data.replace(3, 5, "maja"); break;
            case  6: data.replace(3, 5, "czerwca"); break;
            case  7: data.replace(3, 5, "lipca"); break;
            case  8: data.replace(3, 5, "sierpnia"); break;
            case  9: data.replace(3, 5, "września"); break;
            case 10: data.replace(3, 5, "października"); break;
            case 11: data.replace(3, 5, "listopada"); break;
            case 12: data.replace(3, 5, "grudnia"); break;
        }
        System.out.println(data);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Podobnie jak w rozwiązaniu zadań 13.10 i 13.11, możemy zamiast instrukcji wyboru zastosować tablicę z nazwami miesięcy (w dopełniaczu).

```
private static final String[] mies = {"", "stycznia", "lutego", "marca",
    "kwietnia", "maja", "czerwca", "lipca", "sierpnia", "września",
    "października", "listopada", "grudnia"};
```

Otrzymamy krótszy kod.

```
int mc = Integer.parseInt(str.substring(3, 5));
StringBuilder data = new StringBuilder(str);
data.replace(2, 3, " ").replace(5, 6, " ").replace(3, 5, mies[mc]);
System.out.println(data);
```

b) Drugą część zadania rozwiązujemy w taki sam sposób. Wystarczy zamiast nazw miesięcy (w instrukcji wyboru lub tablicy) wpisać odpowiednie liczby rzymskie: `I`, `II`, `III`, …, `XII`.

Zadanie 13.15. Z13_15.java

Algorytm wyznaczania dnia tygodnia na podstawie daty, podany przez Christiana Zellera, uprościł matematyk Mike Keith. Niezbędne obliczenia możemy zrealizować, korzystając ze wzoru:

$$dt = \left(\left\lfloor \frac{23m}{9} \right\rfloor + d + 4 + y + \left\lfloor \frac{z}{4} \right\rfloor - \left\lfloor \frac{z}{100} \right\rfloor + \left\lfloor \frac{z}{400} \right\rfloor - c \right) \bmod 7$$

W podanym wzorze występują następujące dane:

- d — numer dnia w miesiącu (ang. *day*);
- m — numer miesiąca (ang. *month*): 1 — styczeń, 2 — luty, …, 12 — grudzień;
- y — liczba wyrażająca rok (ang. *year*).

Wykorzystujemy dwie zmienne pomocnicze:

- z — liczba wyrażająca rok z poprawką: $z = y–1$, gdy $m < 3$ (styczeń lub luty);

  $z = y$, gdy $m >= 3$ (od marca do grudnia);
- c — wartość korygująca wynik obliczeń (ang. *correction*): $c = 0$, gdy $m < 3$ (styczeń lub luty);

  $c = 2$, gdy $m >= 3$ (od marca do grudnia).

Wartości tych zmiennych wyznaczymy, używając instrukcji warunkowych lub wyrażeń warunkowych.

W obliczeniach wykorzystujemy dwie funkcje:

- $\lfloor \ \rfloor$ — oznacza część całkowitą liczby (ang. *floor* — zaokrąglenie w dół, podłoga) — odpowiedni wynik otrzymamy, wykonując dzielenie całkowite;
- *mod* — operator modulo (reszta z dzielenia) — w Javie jest to operator `%`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Wyliczoną wartość *dt* (dzień tygodnia) zinterpretujemy następująco: *0* — niedziela, *1* — poniedziałek, *2* — wtorek, *3* — środa, *4* — czwartek, *5* — piątek, *6* — sobota.

```
import java.util.Scanner;
public class Z13_15 {
    public static final String[] days = {"niedziela", "poniedziałek",
       "wtorek", "środa", "czwartek", "piątek", "sobota"};

    public static void main(String args[]) {
        System.out.println("Jaki to dzień tygodnia?");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj datę w formacie dd.mm.rrrr: ");
        String str = input.next();
        /* Zakładamy, że podana data jest poprawna. Odczytujemy dane. */
        int y = Integer.parseInt(str.substring(6, 10));
        int m = Integer.parseInt(str.substring(3, 5));
        int d = Integer.parseInt(str.substring(0, 2));
        /* Wykonujemy obliczenia według wzoru Zellera z poprawkami Keitha. */
        int z = (m < 3)?y-1:y;
        int c = (m < 3)?0:2;
        int dt = (23*m/9+d+4+y+z/4-z/100+z/400-c)%7;
        /* Budujemy odpowiedź i zamieniamy wartość dt na nazwę dnia. */
        StringBuilder data = new StringBuilder(str);
        data.append(" - ").append(days[dt]);
        System.out.println(data);
    }
}
```


Uwaga

Fragment wzoru `z/4-z/100+z/400` musi pozostać w tej postaci. W każdym z dzieleń (całkowitych) „gubiona" jest część ułamkowa, czyli wynik obcinamy do części całkowitej (w dół). Potraktowanie tego wyrażenia jako działania na ułamkach i sprowadzenie do wspólnego mianownika jest błędem!

14. Instrukcja pętli typu do-while

Zadanie 14.1. Z14_1.java

Tworzymy obiekt `input` klasy `Scanner` do wczytywania danych z klawiatury. Deklarujemy dwie zmienne typu `double`: `suma` — do obliczania i przechowywania sumy liczb, `liczba` — do przechowywania wartości aktualnie wprowadzonej liczby. Zmienna `suma` jest zainicjowana wartością początkową sumy (zerem). Pobieranie danych i dodawanie ich do sumy umieszczone jest w ciele pętli typu `do-while` (ze sprawdzaniem warunku na końcu). Jeśli wprowadzona wartość jest różna od zera, to powtarzamy instrukcje wewnątrz pętli, czyli wczytujemy kolejną liczbę z klawiatury, dodajemy ją do sumy oraz ponownie sprawdzamy warunek. Cykl się powtarza. Jeśli wprowadzona liczba jest zerem, to jej dodanie do sumy nie zmieni wyniku (liczba *0* jest elementem neutralnym dodawania), natomiast pętla zostanie zakończona. Na koniec zamkniemy obiekt `input` i wyświetlimy wynik.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
public class Z14_1 {
    public static void main(String args[]) {
        System.out.println("Sumowanie serii liczb zakończonej zerem");
        Scanner input = new Scanner(System.in);
        double suma = 0;
        double liczba;
        do {
            System.out.print("x = ");
            liczba = input.nextDouble();
            suma += liczba;
        } while (liczba != 0);
        input.close();
        System.out.printf("Suma liczb %f\n", suma);
    }
}
```

Zadanie 14.2. Z14_2.java

Metodę pobierającą dane ze strumienia wejściowego umieścimy wewnątrz pętli typu `do-while`. W warunku końcowym porównujemy wprowadzoną liczbę z zerem. Jeśli ta liczba jest równa zeru lub mniejsza (*nie spełnia warunków treści zadania*, ale spełnia warunek `liczba <= 0` na końcu pętli), to powtarzamy cykl, wczytując kolejną wartość. Podanie poprawnej wartości (w tym przypadku liczby dodatniej) powoduje zakończenie pętli.

```
import java.util.Scanner;
public class Z14_2 {
    public static void main(String args[]) {
        System.out.println("Wczytywanie wyłącznie liczby dodatniej");
        Scanner input = new Scanner(System.in);
        double liczba;
        do {
            System.out.print("Podaj liczbę dodatnią, x = ");
            liczba = input.nextDouble();
        } while (liczba <= 0);
        input.close();
        System.out.printf("Wprowadzona liczba %f jest dodatnia.\n", liczba);
    }
}
```

Ten kod możemy stosować wszędzie tam, gdzie potrzebujemy danych spełniających określone warunki początkowe.

Zadanie 14.3. Z14_3.java

Zaczynamy od liczby `n = 7` (jednokrotność liczby *7*). W pętli wyświetlamy wartość liczby, przecinek i odstęp oddzielający tę liczbę od wartości następnej oraz zwiększamy liczbę o *7* (kolejna wielokrotność liczby *7*). Jeśli następna wielokrotność jest mniejsza od *7*, to cykl powtarzamy — gdy wielokrotność przekroczy wartość *100*, kończymy program.

```
public class Z14_3 {
    public static void main(String args[]) {
        System.out.println("Wielokrotności liczby 7 mniejsze od 100: ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        int n = 7;
        do {
            System.out.print(n+", ");
            n += 7;
        } while (n < 100);
        System.out.println();
    }
}
```

Możemy oddzielać liczby innymi znakami, np. odstępem, lub zapisywać każdą w innym wierszu. Jeśli nie podoba nam się przecinek (i niewidoczny odstęp) na końcu ciągu liczb, to możemy nieco zmodyfikować kod i wewnątrz pętli zbadać, jaki znak postawić — przecinek czy kropkę.

```
int n = 7;
do {
    System.out.print(n);
    n += 7;
    if (n < 100)
        System.out.print(", ");
    else
        System.out.print(".");
} while (n < 100);
```

W kolejnym wariancie przecinek i odstęp stawiamy przed liczbą, z wyjątkiem pierwszej wartości, a po ostatniej wielokrotności stawiamy kropkę.

```
int n = 7;
System.out.print(n);
do {
    System.out.print(", ");
    n += 7;
    System.out.print(n);
} while (n < 100);
System.out.println(".");
```

Zadanie 14.4. Z14_4.java

Pomysł przedstawiony w rozwiązaniu zadania 14.2 zastosujemy do wprowadzenia dodatniej wartości promienia koła.

```
import java.util.Scanner;

public class Z14_4 {
    public static void main(String args[]) {
        System.out.println("Obliczanie pola powierzchni koła");
        Scanner input = new Scanner(System.in);
        double r;
        do {
            System.out.print("Podaj promień koła, r = ");
            r = input.nextDouble();
            if (r <= 0)
                System.out.println("Promień koła powinien być liczbą
                    dodatnią!");
        } while (r <= 0);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        input.close();
        System.out.printf("Pole koła P = %f\n", Math.PI*r*r);
    }
}
```

Wartość wyrażenia logicznego `r <= 0` jest obliczana dwukrotnie. Raz w instrukcji warunkowej, po to aby wyświetlić komunikat o błędnie podanej wartości, a drugi raz podczas badania warunku zakończenia pętli.

Nieskończoną pętlę `do {...} while(true)` (warunek zawsze spełniony — `true`) możemy przerwać poleceniem `break`; gdy `r <= 0`, wyświetlamy komunikat o błędzie, w przeciwnym razie przerywamy pętlę (promień jest dodatni).

```
do {
    System.out.print("Podaj promień koła, r = ");
    r = input.nextDouble();
    if (r <= 0)
        System.out.println("Promień koła powinien być liczbą dodatnią!");
    else
        break;
} while (true);
```

Zadanie 14.5. Z14_5.java

Pole pierścienia kołowego obliczymy ze wzoru

$$P = \pi R^2 - \pi r^2 = \pi\left(R^2 - r^2\right) = \pi\left(R - r\right)\left(R + r\right),$$

gdzie $0 < r < R$, r — promień wewnętrzny (w kodzie `r1`), R — promień zewnętrzny (`r2`). Mając do wyboru trzy wzory, wybierzemy ten, który wymaga najmniejszej liczby działań do wykonania — licząc od lewej strony: 4 mnożenia i odejmowanie (5 działań); 3 mnożenia i odejmowanie (4 działania); *2 mnożenia, odejmowanie i dodawanie* (*4 działania*) — w tym najmniej mnożeń, które zabierają więcej czasu niż odejmowanie lub dodawanie.

```
import java.util.Scanner;
public class Z14_5 {
    public static void main(String args[]) {
        System.out.println("Obliczanie pola pierścienia kołowego");
        Scanner input = new Scanner(System.in);
        double r1, r2;
        do {
            System.out.print("Podaj promień zewnętrzny, R = ");
            r2 = input.nextDouble();
            if (r2 <= 0)
                System.out.println("Promień powinien być dodatni!");
        } while (r2 <= 0);
        do {
            System.out.print("Podaj promień wewnętrzny, r = ");
            r1 = input.nextDouble();
            if (r1 <= 0)
                System.out.println("Promień powinien być dodatni!");
            if (r1 >= r2)
                System.out.println("Promień wewnętrzny powinien być mniejszy
                                    od zewnętrznego!");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        } while (r1 <= 0 || r1 >= r2);
        input.close();
        System.out.printf("Pole pierścienia P = %f\n", Math.PI*(r2-r1)*(r2+r1));
    }
}
```

Zwróćmy uwagę na oddzielne badanie warunków wewnątrz drugiej pętli (wskazanie przyczyny błędu) i alternatywę warunków powodujących powtórne wczytywanie długości promienia wewnętrznego.

Zadanie 14.6. Z14_6.java

Tworzymy trzy zmienne `a`, `b` i `c` typu `int` do przechowywania trzech kolejnych wartości ciągu Fibonacciego. Zaczynamy (od początku ciągu), podstawiając `a = 1`, `b = 1` i obliczając trzeci wyraz `c = a+b` (wartość początkowa *2*). Wyświetlamy dwa początkowe wyrazy `a` i `b`. Następnie wchodzimy do pętli:

- wyświetlamy wyraz `c`,
- przesuwamy dwie wartości `a = b; b = c;`,
- obliczamy kolejny wyraz ciągu `c = a+b`.

Gdy wartość `c` jest mniejsza niż *1000*, powtarzamy cykl.

```
public class Z14_6 {
    public static void main(String args[]) {
        System.out.println("Liczby Fibonacciego mniejsze od 1000:");
        int a = 1, b = 1; // Dwa pierwsze wyrazy ciągu
        System.out.print(a+", "+b);
        int c = a+b;      // Kolejny wyraz ciągu
        do {
            System.out.print(", "+c);
            a = b;
            b = c;
            c = a+b;
        } while (c < 1000);
        System.out.println(".");
    }
}
```

Zwróćmy uwagę na wyświetlanie przecinka i odstępu między liczbami oraz postawmy kropkę na końcu odpowiedzi (po wyjściu z pętli).

Zadanie 14.7. Z14_7.java

Tym razem nie wyświetlamy liczb Fibonacciego, ale obliczamy, ile jest w ciągu liczb mniejszych od zadanej liczby całkowitej *n*. Ponieważ najmniejszą liczbą w tym ciągu jest 1, to dla $n \leq 1$ poprawną odpowiedzią będzie 0 i taką wartość początkową nadajemy zmiennej `licznik`. W trakcie obliczeń ta wartość może się zmienić. Dla $n = 2$ mamy w ciągu dwie wartości mniejsze, zmiennej `licznik` nadamy wartość *2*. Jeśli $n > 2$, to obliczamy w pętli kolejne wyrazy ciągu (według algorytmu z zadania 14.6) i inkrementujemy zmienną `licznik`. Pętla kończy się, gdy kolejny obliczony wyraz `c` osiągnie lub przekroczy wartość `n` (dla `c < n` cykl jest kontynuowany).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
public class Z14_7 {
    public static void main(String args[]) {
        System.out.println("Ilość liczb Fibonacciego mniejszych od n");
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj liczbę całkowitą, n = ");
        int n = input.nextInt();
        input.close();
        int licznik = 0;
        if (n == 2)
            licznik = 2;
        else if (n > 2) {
            int a = 1, b = 1;
            int c = a+b;
            licznik += 2;  // dwa początkowe wyrazy
            do {
                ++licznik; // kolejny wyraz ciągu
                a = b;
                b = c;
                c = b+a;
            } while (c < n);
        }
        System.out.println(licznik);
    }
}
```

Zadanie 14.8. Z14_8.java

Obliczenia powtarzamy w pętli, której wykonywanie zależy od wartości zmiennej `a`:

```
double a;
do {
    /* Wprowadzanie danych, badanie poprawności i obliczenia */
} while(a > 0);
```

Pętla wykona się co najmniej raz, a jej dalsze działanie zależy od pobranej (w instrukcji wewnątrz pętli) wartości zmiennej `a`. Dodatnia wartość zmiennej (decyduje wynik porównania `a > 0`) spowoduje powtórzenie cyklu, ujemna lub zero — zakończenie pętli.

Druga, wewnętrzna pętla ma za zadanie testowanie poprawności danych i zmusza użytkownika do wprowadzenia liczby dodatniej (prawidłowa długość boku) lub 0 — przyjęty umownie sygnał zakończenia obliczeń. Obliczenia są wykonywane wyłącznie, gdy `a > 0`, natomiast po podaniu wartości `0` pojawi się komunikat `Koniec obliczeń` i program zakończy działanie.

```
import java.util.Scanner;
public class Z14_8 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        double a;
        do {
            System.out.println("Pole trójkąta równobocznego");
            do {
                System.out.print("Długość boku (0 - koniec), a = ");
                a = input.nextDouble();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            } while (a < 0);
            if (a > 0) {
                double pole = a*a*Math.sqrt(3)/4;
                System.out.println("P = "+pole);
            } else
                System.out.println("Koniec obliczeń.");
        } while (a > 0);
        input.close();
    }
}
```

Wprowadźmy kilka poprawek. Zamiast wielokrotnego wywoływania funkcji `sqrt` dla stałego argumentu `3` i dzielenia wyniku przez `4` wprowadzimy dodatkową zmienną (współczynnik występujący we wzorze na pole trójkąta równobocznego) `double wsp = Math.sqrt(3)/4` przechowującą przybliżenie liczby $\frac{\sqrt{3}}{4}$. Podanie ujemnej wartości dla zmiennej `a` zasygnalizujemy komunikatem `Długość powinna być dodatnia!` Główną pętlę programu przerwiemy po wprowadzeniu długości boku równej `0`, stosując instrukcję `break`, więc końcowy warunek możemy zastąpić wartością stałą `true` (pętla będzie nieskończona).

```
double a;
double wsp = Math.sqrt(3)/4;
do {
    System.out.println("Pole trójkąta równobocznego");
    do {
        System.out.print("Długość boku (0 - koniec), a = ");
        a = input.nextDouble();
        if (a < 0)
            System.out.println("Długość powinna być dodatnia!");
    } while (a < 0);
    if (a > 0) {
        double pole = a*a*wsp;
        System.out.printf("P = %.2f\n", pole);
    } else {
        System.out.println("Koniec obliczeń.");
        break;
    }
} while (true);
```

Zadanie 14.9. Z14_9.java

Zmienna `odp` typu `char` przechowuje informację potrzebną do podjęcia decyzji o powtórzeniu obliczeń. Na początku nadajemy jej dowolną wartość różną od `'t'` i `'n'`.

Główna pętla zawiera obliczenia (zgodnie z treścią zadania) i pytanie o powtarzanie obliczeń, modyfikujące wartość zmiennej `odp` stosownie do podjętej decyzji.

```
char odp = "?";
do {
    /* instrukcje - obliczenia, zapytanie o powtórzenie obliczeń */
} while(odp == 't');
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Przed główną pętlą tworzymy obiekt `input` klasy `Scanner`, a po jej zakończeniu zamykamy go. Stosując metody obiektu `input`, odczytamy dane potrzebne do obliczeń oraz odpowiedź na pytanie, czy kontynuujemy obliczenia.

Pytanie o kontynuację obliczeń zawrzemy w kolejnej pętli. Przed wejściem do pętli odczytujemy cały wiersz, usuwając w ten sposób resztę znaków ze strumienia (aby nie miały wpływu na podejmowanie decyzji). W pętli zadajemy użytkownikowi pytanie `Czy obliczamy dalej (t/n)?` z sugerowaną odpowiedzią `t` — tak lub `n` — nie. Każda inna odpowiedź będzie ignorowana i spowoduje powtórzenie pytania (również naciśnięcie klawisza *Enter* — pusty łańcuch). Jeśli podaną odpowiedzią (zmienna `s`) nie jest pusty łańcuch, to pobieramy pierwszy znak do zmiennej `odp`, a resztę znaków pomijamy.

```
String s = input.nextLine();
do {
    System.out.print("Czy obliczamy dalej (t/n)? ");
    s = input.nextLine();
    if (!s.isEmpty())
        odp = s.charAt(0);
} while(s.isEmpty() || odp != 't' && odp != 'n');
```

Po wyjściu z pętli zmienna `odp` może mieć wyłącznie wartość `'t'` lub `'n'`. Ta odpowiedź zadecyduje o dalszym działaniu programu. Warunek `s.isEmpty() || odp != 't' && odp != 'n'` możemy zastąpić równoważnym warunkiem `s.isEmpty() || !(odp == 't' || odp == 'n')`.

Oto kompletny listing programu:

```
import java.util.Scanner;
public class Z14_9 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        char odp = '?';
        do {
            /* Obliczenia */
            System.out.println("Obliczanie długości okręgu");
            double r;
            do {
                System.out.print("Podaj promień okręgu, r = ");
                r = input.nextDouble();
            } while (r <= 0);
            System.out.printf("Długość okręgu L = %.4f\n", 2*Math.PI*r);
            /* Koniec obliczeń * Pytanie o kontynuację obliczeń */
            String s = input.nextLine();
            do {
                System.out.print("Czy obliczamy dalej (t/n)? ");
                s = input.nextLine();
                if (!s.isEmpty())
                    odp = s.charAt(0);
            } while(s.isEmpty() || odp != 't' && odp != 'n');
        } while (odp == 't');
        input.close();
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

15. Instrukcja pętli typu while

Zadanie 15.1. Z15_1.java

Dane wejściowe są sprawdzane podczas ich wprowadzania (nie dopuszczamy liczb całkowitych ujemnych) i proces dodawania przeprowadzimy dla liczb naturalnych (całkowitych nieujemnych). Algorytm obliczeń jest następujący: dopóki liczba `n` jest dodatnia, zmniejszamy `n` o `1` (`--n`) i jednocześnie zwiększamy `m` o `1` (`++m`):

```
while (n > 0) {
    --n; // Zmniejszamy n o 1.
    ++m; // Zwiększamy m o 1.
}
```

Po wyjściu z pętli zmienna `n` ma wartość `0`, a `m` zawiera obliczoną sumę. Obrazowo ten algorytm możemy przedstawić jako przekładanie patyczków z jednego stosu na drugi, oczywiście dopóki stos, z którego zabieramy patyczki, nie jest pusty.

```
import java.util.Scanner;
public class Z15_1 {
    public static void main(String args[]) {
        /* Wprowadzanie danych */
        System.out.println("Podaj dwie liczby naturalne:");
        Scanner input = new Scanner(System.in);
        int m, n;
        do {
            System.out.print("m = ");
            m = input.nextInt();
        } while (m < 0);
        do {
            System.out.print("n = ");
            n = input.nextInt();
        } while (n < 0);
        input.close();
        /* Obliczenia i wynik */
        while (n > 0) {
            --n; // Zmniejszamy n o 1.
            ++m; // Zwiększamy m o 1.
        }
        System.out.printf("Suma liczb %d\n", m);
    }
}
```

Pętlę realizującą obliczenia możemy zapisać również tak: `while (n-- > 0) ++m;`. Zastosowana postdekrementacja (`n--`) w wyrażeniu `n-- > 0` powoduje skrócenie zapisu. Działa to w taki sposób: tworzona jest kopia wartości zmiennej `n`, wartość zmiennej `n` jest zmniejszana o `1`, do porównywania z zerem brana jest przechowana kopia. Nie wpływa to na liczbę powtórzeń pętli, więc program działa tak samo. Pozostaje jednak pewna różnica w wartości końcowej zmiennej `n`. W tym programie nie ma to jednak znaczenia.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 15.2. Z15_2.java

Jeśli liczba n jest dodatnia, to realizujemy algorytm omówiony w rozwiązaniu zadania 15.1. Dla ujemnych wartości n musimy postąpić odwrotnie — będziemy zwiększać n o 1 i jednocześnie zmniejszać m o 1. Gdy n osiągnie wartość 0, to m będzie równe sumie początkowych wartości m i n.

W zależności od znaku zmiennej n wykona się albo pierwsza pętla (n > 0), albo druga (n < 0), albo żadna z nich (n = 0). Za każdym razem końcowa wartość m będzie szukaną sumą.

```
import java.util.Scanner;
public class Z15_2 {
    public static void main(String args[]) {
        /* Wprowadzanie danych */
        System.out.println("Podaj dwie liczby całkowite:");
        Scanner input = new Scanner(System.in);
        System.out.print("m = ");
        int m = input.nextInt();
        System.out.print("n = ");
        int n = input.nextInt();
        input.close();
        /* Obliczenia i wynik */
        while (n > 0) {
            --n;
            ++m;
        }
        while (n < 0) {
            ++n;
            --m;
        }
        System.out.printf("Suma liczb %d\n", m);
    }
}
```

Upraszczając zapis obu pętli, nie uzyskamy oczekiwanego rezultatu. Wynik będzie zawsze o 1 mniejszy od sumy liczb. Jest to efekt zmiany wartości zmiennej n podczas badania warunku, na co zwracano uwagę w rozwiązaniu zadania 15.1.

```
while (n-- > 0) ++m;
while (n++ < 0) --m;
```

Możemy to skorygować, zwiększając końcową wartość m o 1 (++m) lub stosując dodatkową instrukcję warunkową:

```
if (n > 0)
    while (n-- > 0) ++m;
else
    while (n++ < 0) --m;
```

Jeśli zmienimy kolejność pętli, to wynik będzie o 1 większy od sumy.

```
while (n++ < 0) --m;
while (n-- > 0) ++m;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Należy wystrzegać się tego typu uproszczeń, a jeśli już się na nie zdecydujemy, to musimy dokładnie analizować działanie kodu. Upraszczając zapis, czynimy kod mniej czytelnym.

Zadanie 15.3a. Z15_3a.java

Dane wprowadzimy podobnie jak w rozwiązaniu zadania 15.1. Skorzystamy z faktu, że mnożenie jest skróconym sposobem zapisu wielokrotnego dodawania tej samej liczby. Ustalimy początkową wartość iloczynu równą `0`. Do tej wartości dodamy `n`-krotnie liczbę `m`, czyli zmienna `n` odegra rolę licznika dodawań.

```
int iloczyn = 0;
while (n > 0) {
    --n;                    // Zmniejszamy (licznik dodawań) n o 1.
    iloczyn = iloczyn+m; // Zwiększamy iloczyn o m.
}
System.out.printf("Iloczyn liczb %d\n", iloczyn);
```

Zadanie 15.3b. Z15_3b.java

Dane wprowadzimy podobnie jak w rozwiązaniu zadania 15.2. Najpierw zgodnie z zasadami arytmetyki ustalimy znak iloczynu, a następnie wykonamy obliczenia dla liczb dodatnich — bezwzględnych wartości z danych wejściowych. Gdy co najmniej jeden z czynników jest zerem, iloczyn będzie równy `0` (wartość początkowa zmiennej `iloczyn`).

```
int iloczyn = 0;
int znak;
if (n != 0 && m != 0) {
    if (n > 0)
        znak = 1;
    else {
        znak = -1;
        n = -n;
    }
    if (m < 0) {
        znak = -znak;
        m = -m;
    }
    while (n > 0) {
        --n;                    // Zmniejszamy n o 1.
        iloczyn = iloczyn+m; // Zwiększamy iloczyn o m.
    }
    if (znak == -1)
        iloczyn = -iloczyn;
}
System.out.printf("Iloczyn liczb %d\n", iloczyn);
```

Zadanie 15.4. Z15_4.java

Dane wprowadzimy podobnie jak w rozwiązaniu zadania 15.2. Dla liczb dodatnich obliczenie ilorazu całkowitego polega na wielokrotnym odejmowaniu dzielnika (dopóki dzielna nie jest mniejsza od dzielnika) i zliczaniu liczby odejmowań. Rolę licznika odgrywa zmienna `iloraz`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
int iloraz = 0;
while (m >= n) {
    ++iloraz;
    m = m-n;
}
```

Jeśli dzielna lub dzielnik są liczbami ujemnymi, ustalimy znak ilorazu (podobnie jak znak iloczynu w zadaniu 15.3b), a obliczenia wykonamy dla bezwzględnych wartości dzielnej i dzielnika. Pozostaje jeszcze problem dzielenia przez 0 — w takiej sytuacji zgłosimy wyjątek `throw new ArithmeticException("dzielenie przez zero")`.

```
/* Obliczenia i wynik */
if (n == 0) {
    throw new ArithmeticException("dzielenie przez zero");
} else {
    int iloraz = 0;
    int znak = 1;
    if (m < 0) {
        znak = -1;
        m = -m;
    }
    if (n < 0) {
        znak = -znak;
        n = -n;
    }
    while (m >= n) {
        ++iloraz;
        m = m-n;
    }
    if (znak == -1)
        iloraz = -iloraz;
    System.out.printf("Iloraz całkowity %d\n", iloraz);
}
```

Zadanie 15.5. Z15_5.java

Dla dodatnich liczb całkowitych `m` i `n` resztę z dzielenia `m` przez `n` wyznaczymy, stosując pętlę `while (m >= n) m = m-n;` — po wyjściu z pętli zmienna `m` ma wartość szukanej reszty. W ogólnym przypadku wyznaczamy znak reszty, który jest taki sam jak znak dzielnej, i wykonujemy obliczenia dla bezwzględnych wartości dzielnej i dzielnika.

```
/* Obliczenia i wynik */
if (n == 0) {
    throw new ArithmeticException("dzielenie przez zero");
} else {
    int znak = 1;
    if (m < 0) {
        znak = -1;
        m = -m;
    }
    if (n < 0)
        n = -n;
    while (m >= n)
        m = m-n;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    if (znak == -1)
        m = -m;
    System.out.printf("Reszta z dzielenia %d\n", m);
}
```

Zadanie 15.6. Z15_6.java

Zaczynamy obliczenia od wartości `p = 0`. Dopóki kwadrat zmiennej `p` nie przekroczy wartości `n` (`p*p <= n`), zwiększamy wartość `p` o `1` (`++p`). Po wyjściu z pętli zmienna `p` jest o `1` większa od pierwiastka całkowitego z `n`, więc korygujemy wynik (`--p`).

```
import java.util.Scanner;
public class Z15_6 {
    public static void main(String args[]) {
        /* Wprowadzanie danych */
        System.out.println("Całkowity pierwiastek kwadratowy");
        Scanner input = new Scanner(System.in);
        int n;
        do {
            System.out.print("Podaj liczbę naturalną, n = ");
            n = input.nextInt();
        } while (n < 0);
        int p = 0;
        while (p*p <= n) ++p;
        --p; // Korekta wyniku
        System.out.printf("Pierwiastek całkowity z %d jest równy %d.\n", n, p);
    }
}
```

Zadanie 15.7. Z15_7.java

Wczytywanie danych zorganizujemy tak, jak w rozwiązaniu zadania 15.1. Zmiennej `nww` (*najmniejsza wspólna wielokrotność*) nadamy wartość początkową równą wartości zmiennej `m` (pierwsza liczba). Dopóki `nww` nie jest podzielne przez drugą liczbę `n` (reszta `nww%n` jest różna od `0`), obliczamy kolejną wielokrotność liczby `m` (`nww += m`). Po wyjściu z pętli zmienna `nww` zawiera poszukiwaną wartość.

```
int nww = m;
while (nww%n != 0)
    nww += m;
System.out.printf("NWW(m, n) = %d.\n", nww);
```

Algorytm jest skuteczny, ale nie zawsze najszybszy. Np. dla pary liczb `m = 1000` i `n = 2` pętla nie wykona się ani razu, a wynikiem będzie liczba `1000`. Natomiast dla pary `m = 2` i `n = 1000` pętla wykona się *499* razy (*1000:2–1*), zanim zmienna `nww` osiągnie prawidłową wartość. Wniosek nasuwa się sam — jako początkową wartość `nww` wybierzmy większą z liczb.

```
int nww;
if (m > n) {
    nww = m;
    while (nww%n != 0)
        nww += m;
} else if (n > m) {
    nww = n;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    while (nww%m != 0)
        nww += m;
} else
    nww = m; // m i n są równe
System.out.printf("NWW(m, n) = %d.\n", nww);
```

Liczba powtórzeń pętli w najgorszym przypadku osiągnie wartość mniejszej z tych liczb, pomniejszoną o 1. W tym przypadku *NWW(m, n) = mn* (liczby *m* i *n* są względnie pierwsze).

Zadanie 15.8. Z15_8.java

Algorytm Euklidesa opiera się na spostrzeżeniu, że dla dowolnych dwóch liczb naturalnych mniejsza spośród nich i ich różnica mają taki sam największy wspólny dzielnik (*NWD*) jak te liczby. Tworzymy zatem ciąg kolejnych par liczb — mniejsza liczba i różnica liczb (od większej odejmujemy mniejszą). Jeśli w wyniku kolejnego odejmowania otrzymamy parę równych liczb, to właśnie ona będzie poszukiwanym *NWD*.

```
while (m != n)
    if (m > n)
        m = m-n;
    else
        n = n-m;
System.out.printf("NWD(m, n) = %d.\n", m);
```

Istnieje druga wersja (szybsza) tego algorytmu. Zauważmy, że wielokrotne odejmowanie tej samej liczby prowadzi do wyznaczenia reszty z dzielenia (zob. rozwiązanie zadania 15.5). Ponadto jeśli uzyskamy resztę równą 0, to dzielnik jest podzielnikiem dzielnej i będzie to właśnie szukany *NWD*. Zobaczmy to na przykładzie:

1. Wersja algorytmu z odejmowaniem — *NWD(32, 10) = NWD(22, 10) = NWD(12, 10) = NWD(2, 10) = NWD(2, 8) = NWD(2, 6) = NWD(2, 4) = NWD(2, 2) = 2*.
2. Wersja z dzieleniem z resztą — *NWD(32, 10) = NWD(10, 2) = **2**, bo 32:10 = 3 r. 2, 10:**2** = 5 r. **0***.

Wprowadzamy pomocniczą zmienną `p`, która jest wykorzystywana do zamiany miejscami wartości przechowywanych w zmiennych `m` i `n`.

```
int p;
while (n != 0) {
    p = m%n;
    m = n;
    n = p;
}
System.out.printf("NWD(m, n) = %d.\n", m);
```

Zadanie 15.9. Z15_9.java

Zgodnie z opisem tworzymy dwie zmienne `wm` i `wn` typu `int`, w których będziemy przechowywali wielokrotności liczb `m` i `n`. Na początku będą to jednokrotności danych wejściowych. Dopóki wielokrotności `wm` i `wn` są różne, obliczmy kolejne krotności tej liczby, której aktualna wielokrotność jest mniejsza od drugiej wielokrotności. Gdy

Ebookpoint.pl kopia dla: pawher@o2.pl

wielokrotności zrównają się, wychodzimy z pętli — najmniejsza wspólna wielokrotność `m` i `n` została obliczona.

```
int wm = m; // wielokrotność liczby m
int wn = n; // wielokrotność liczby n
while (wm != wn)
    if (wm > wn)
        wn += n; // kolejna wielokrotność liczby n
    else
        wm += m; // kolejna wielokrotność liczby m
System.out.printf("NWW(m, n) = %d.\n", wm);
```

16. Instrukcja pętli typu for

Zadanie 16.1. Z16_1.java

W rozwiązaniu zadania wykorzystamy wzór $1+3+...+(2n-1)=n^2$. Sumowanie przeprowadzimy w pętli `for` dla *i = 0*, *1*, *2*, … *n-1* — czyli w pętli dodamy *n* początkowych liczb nieparzystych, uzyskując sumę n^2.

```
int kw = 0;   // kwadrat liczby 0, początkowa wartość sumy
int lnp = 1; // pierwsza liczba nieparzysta
for(int i = 0; i < n; ++i) {
    kw += lnp; // kolejny kwadrat (dodana kolejna liczba nieparzysta)
    lnp += 2;   // kolejna liczba nieparzysta
}
System.out.printf("Kwadrat liczby %d jest równy %d.\n", n, kw);
```

Przeanalizujmy, jak działa pętla `for(int i = 0; i < n; ++i) { instrukcja }`. Po słowie kluczowym `for` w nawiasach okrągłych mamy trzy ważne elementy, oddzielone średnikami (`;`). Pierwszy element to *instrukcja inicjująca*. Przed wejściem do pętli deklarowana i inicjowana jest zmienna `i` (`int i = 0`), która będzie odgrywała rolę licznika powtórzeń pętli. Przed każdym wykonaniem *instrukcji* zawartej w ciele pętli (`{ instrukcja }`) sprawdzany jest *warunek* (`i < n`) — jeśli warunek jest spełniony (wartość `true`), to instrukcja jest wykonywana, w przeciwnym wypadku (wartość `false`) następuje zakończenie pętli. Po wykonaniu *instrukcji* z ciała pętli wykonywana jest *instrukcja modyfikująca* — w naszym przykładzie inkrementująca licznik powtórzeń pętli (`++i`).

Używając operatora przecinkowego (`,`) do grupowania wyrażeń, możemy przenieść deklarację i zainicjowanie zmiennej `lnp` do *instrukcji inicjującej* pętlę `for`. Podobnie zwiększanie tej zmiennej o *2* umieścimy w instrukcji modyfikującej. W ciele pętli pozostanie instrukcja podstawiania (`kw += lnp`) sumująca liczby nieparzyste.

```
int kw = 0;
for(int i = 0, lnp = 1; i < n; ++i, lnp += 2)
    kw += lnp;
```

Ponieważ zmienne zainicjowane w pętli `for` mają zasięg lokalny, ograniczony do ciała tej pętli, to deklaracja i inicjalizacja zmiennej `kw` muszą nastąpić przed pętlą. Nic nie

Ebookpoint.pl kopia dla: pawher@o2.pl

stoi na przeszkodzie, aby instrukcję `kw += lnp;` umieścić w *instrukcji modyfikującej* pętli `for`.

```
int kw = 0;
for(int i = 0, lnp = 1; i < n; ++i, kw += lnp, lnp += 2);
```

Instrukcja w ciele pętli jest instrukcją pustą (nie zapomnijmy o średniku), natomiast instrukcje tworzące *instrukcję modyfikującą* muszą być ustawione w odpowiedniej kolejności. Takich „uproszczeń" jednak nie polecamy.

Zadanie 16.2. Z16_2.java

Ze standardowego strumienia wejściowego wczytujemy jedno słowo (łańcuch `s`). Przeglądając w pętli `for` (znak po znaku) łańcuch `s`, budujemy obiekt `str` (klasy `StringBuilder`) ze znaków łańcucha `s`. Ponieważ kolejne znaki wstawiamy na początek łańcucha, to w efekcie otrzymamy odwrócony ciąg znaków. Obiekt `str` zamienimy na łańcuch znaków (obiekt klasy `String`), zapisany małymi literami (`str.toString().toLowerCase()`) i porównamy z łańcuchem wejściowym `s` zapisanym małymi literami (`s.toLowerCase()`). Porównanie to zrealizujemy, używając metody `equals()` dla jednego z tych łańcuchów, natomiast drugi łańcuch będzie parametrem. Tę samą wartość otrzymamy dla porównania `s.toLowerCase().equals(str.toString().toLowerCase())` oraz `str.toString().toLowerCase().equals(s.toLowerCase())`. W zależności od wyniku porównania formułujemy odpowiedź.

```
import java.util.Scanner;
public class Z16_2 {
    public static void main(String args[]) {
        System.out.println("Czy to jest palindrom?");
        /* Wczytywanie danych */
        Scanner input = new Scanner(System.in);
        System.out.print("Podaj słowo do sprawdzenia: ");
        String s;
        s = input.next();
        input.close();
        /* Odwracanie słowa */
        int n = s.length();
        StringBuilder str = new StringBuilder();
        for(int i = 0; i < n; ++i)
            str.insert(0, s.charAt(i));
        /* Porównywanie słowa podanego ze słowem odwróconym */
        if (s.toLowerCase().equals(str.toString().toLowerCase()))
            System.out.println("Słowo: \""+s+"\" - jest palindromem.");
        else
            System.out.println("Słowo: \""+s+"\" - nie jest palindromem.");
    }
}
```

Odwrócenie znaków możemy zrealizować, czytając znaki łańcucha `s` od końca. W tym celu użyjemy pętli `for` z odliczaniem w dół.

```
int n = s.length();
StringBuilder str = new StringBuilder();
for(int i = n-1; i >= 0; --i)
    str.append(s.charAt(i));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Korzystając z dostępnych metod w klasach `String` i `StringBuilder`, możemy zrealizować zadanie bez pętli i deklarowania pomocniczych zmiennych.

```
String s = input.next();
if (s.toLowerCase().equals(new
StringBuilder(s).reverse().toString().toLowerCase()))
    System.out.println("Słowo: \""+s+"\" - jest palindromem.");
else
    System.out.println("Słowo: \""+s+"\" - nie jest palindromem.");
```

Zadanie 16.3. Z16_3.java

Ze standardowego strumienia wejściowego wczytujemy jeden wiersz (łańcuch `s`). Przeglądając w pętli `for` (znak po znaku) łańcuch `s`, budujemy obiekt `str` (klasy `StringBuilder`) ze znaków łańcucha `s`, pomijając przy tym znaki odstępu:

```
s = input.nextLine();
int n = s.length();
StringBuilder str = new StringBuilder();
/* Budujemy odwrócony łańcuch, bez odstępów. */
for(int i = 0; i < n; ++i)
    if (s.charAt(i) != ' ')
        str.insert(0, s.charAt(i));
```

lub

```
for(int i = n-1; i >= 0; --i)
    if (s.charAt(i) != ' ')
        str.append(s.charAt(i));
```

Łańcuch `s` może zawierać odstępy, a w odwróconym łańcuchu w obiekcie `str` odstępów nie ma, więc porównywanie tych łańcuchów nie da prawidłowego rezultatu. Możemy porównać łańcuch `str.toString().toLowerCase()` z łańcuchem odwróconym (metodą `reverse()`) `str.reverse().toString().toLowerCase()`. Należy pamiętać, że wywołanie metody `str.reverse()` przestawi elementy w łańcuchu `str`.

Proponujemy zrealizować badanie *palindromu* w pętli:

```
boolean palindrom = true;
n = str.length();
for(int i = 0, j = n-1; i < j; ++i, --j)
    palindrom = palindrom && Character.toLowerCase(str.charAt(i)) ==
Character.toLowerCase(str.charAt(j));
```

Zakładamy, że wyrażenie jest palindromem (zmienna logiczna `palindrom = true`). Początkowo w pętli zmienna sterująca `i` wskazuje na pierwszy znak łańcucha, a zmienna `j` na ostatni znak. Powtarzanie cyklu porównań trwa, jeśli `i < j`. Po każdym cyklu `i` zwiększa wartość o `1` (`++i`), a `j` zmniejsza wartość o `1` (`--j`) — porównywane znaki zbliżają się do środka łańcucha. Porównywane znaki zamieniamy na małe litery, a koniunkcję wyniku porównania i zmiennej `palindrom` przypisujemy do zmiennej `palindrom`. Jeśli zmienna `palindrom` przyjmie wartość `false` w którymś porównaniu, to tak już pozostanie do końca pętli. Dlatego też moglibyśmy w tym momencie porównywanie zakończyć — wystarczy zmodyfikować warunek w pętli `for(int i = 0, j = n-1; i < j && palindrom; ++i, --j)`.

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Aby uprościć wyrażenia porównujące znaki, możemy w trakcie budowania łańcucha `str` zamieniać litery na małe.

Zadanie 16.4. Z16_4.java

Tabliczkę mnożenia zbudujemy w postaci dwuwymiarowej tabeli z wierszem nagłówka, zawierającym symbol mnożenia (wykorzystano w tym celu wielką literę *X*), i liczbami od *1* do *9* (dziewięć kolumn). Boczek tabeli zawiera również liczby od 1 do 9, odpowiadające dziewięciu wierszom tabliczki mnożenia. W komórkach, na skrzyżowaniu wiersza i kolumny, wpisujemy odpowiednie iloczyny. Pętlę `for` stosujemy czterokrotnie:

- do zbudowania łańcucha znaków `linia`, zawierającego *51* znaków minus (`-`) i używanego do rysowania poziomych linii;
- do zbudowania wiersza nagłówka;
- dwie zagnieżdżone pętle wypełniają tabelę — pętla zewnętrzna tworzy kolejne wiersze, pętla wewnętrzna wypełnia je iloczynami.

```
public class Z16_4 {
    public static void main(String args[]) {
        System.out.println("Tabliczka mnożenia\n");
        /* Linia pozioma */
        StringBuilder linia = new StringBuilder();
        for(int i = 0; i < 51; ++i)
            linia.append("-");
        /* Wiersz nagłówka */
        System.out.println(linia);
        System.out.print("|  X |");
        for(int i = 1; i < 10; ++i)
            System.out.printf("%3d |", i);
        System.out.println();
        System.out.println(linia);
        /* Tabliczka mnożenia */
        for(int j = 1; j < 10; ++j) {
            System.out.printf("|%3d |", j);
            for(int i = 1; i < 10; ++i)
                System.out.printf("%3d |", i*j);
            System.out.println();
        }
        System.out.println(linia);
    }
}
```

Zadanie 16.5. Z16_5.java

W rozwiązaniu zadania zastosujemy schemat opracowany w zadaniu 16.4. Tabelki mają mniejszy wymiar, zawierają (nie licząc nagłówka i boczku) po *6* kolumn i *6* wierszy odpowiadających liczbom od *0* do *10* w układzie piątkowym, czyli od 0 do 5 w układzie dziesiątkowym. Obliczenia wykonujemy na liczbach całkowitych zapisanych w układzie dziesiątkowym, a wyniki konwertujemy na układ piątkowy (`Integer.toString(i, 5)`).

Ebookpoint.pl kopia dla: pawher@o2.pl

Podobny kod wykorzystujemy dwukrotnie, budując tabliczkę dodawania i tabliczkę mnożenia.

```
public class Z16_5 {
    public static void main(String args[]) {
        System.out.println("Piątkowy system liczenia\n");
        StringBuilder linia = new StringBuilder();
        for(int i = 0; i < 36; ++i)
            linia.append("-");
        /* Tabliczka dodawania */
        System.out.println("Tabliczka dodawania");
        System.out.println(linia);
        System.out.print("|  X |");
        for(int i = 0; i <= 5; ++i)
            System.out.printf("%3s |", Integer.toString(i, 5));
        System.out.println();
        System.out.println(linia);
        for(int j = 0; j <= 5; ++j) {
            System.out.printf("|%3s |", Integer.toString(j, 5));
            for(int i = 0; i <= 5; ++i)
                System.out.printf("%3s |", Integer.toString(i+j, 5));
            System.out.println();
        }
        System.out.println(linia);

        /* Tabliczka mnożenia */
        System.out.println("Tabliczka mnożenia");
        System.out.println(linia);
        System.out.print("|  X |");
        for(int i = 0; i <= 5; ++i)
            System.out.printf("%3s |", Integer.toString(i, 5));
        System.out.println();
        System.out.println(linia);
        for(int j = 0; j <= 5; ++j) {
            System.out.printf("|%3s |", Integer.toString(j, 5));
            for(int i = 0; i <= 5; ++i)
                System.out.printf("%3s |", Integer.toString(i*j, 5));
            System.out.println();
        }
        System.out.println(linia);
    }
}
```

Zadanie 16.6. Z16_6.java

Ciąg harmoniczny to ciąg odwrotności kolejnych liczb naturalnych $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$. Najpierw wprowadzamy liczbę wyrazów ciągu `n` — robimy to w pętli (`do ... while ...`), nie pozwalając użytkownikowi na podanie zera lub liczby ujemnej (w takim przypadku ponawiamy prośbę o podanie liczby).

Znając liczbę wyrazów ciągu, w pętli (`for`) obliczamy kolejne wyrazy i dodajemy je do sumy (`suma += 1.0/i` — aby uzyskać wynik zmiennoprzecinkowy, dzielna lub dzielnik musi być liczbą zmiennoprzecinkową. Alternatywą dla podanej instrukcji jest instrukcja `suma += 1/(double)i`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
public class Z16_6 {
    public static void main(String args[]) {
        System.out.println("Suma wyrazów ciągu harmonicznego");
        Scanner input = new Scanner(System.in);
        int n;
        do {
            System.out.print("Liczba wyrazów, n = ");
            n = input.nextInt();
        } while (n <= 0);
        input.close();
        double suma = 0.0;
        for(int i = 1; i <= n; ++i)
            suma += 1.0/i;
        System.out.printf("Suma %d wyrazów ciągu harmonicznego jest
            równa %f.\n", n, suma);
    }
}
```

Zadanie 16.7. Z16_7.java

Wartość liczby `n` użytkownik podaje z klawiatury. Podobnie jak w rozwiązaniu zadania 16.6, ograniczamy wprowadzane wartości do liczb naturalnych (całkowitych nieujemnych). Ponieważ *0! = 1* i *1! = 1*, to ustalamy początkową wartość zmiennej `silnia` równą *1* (`long silnia = 1L;` — litera `L` lub `l` na końcu literału oznacza, że jest to wielkość typu `long`. Mała litera `l` może mylić się z cyfrą `1`, zatem lepiej stosować wielką literę `L`). W pętli (`for`) mnożymy wartość zmiennej `silnia` przez kolejne liczby naturalne, uzyskując ostateczny wynik iloczynu $1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, czyli wartość *n!*.

```
import java.util.Scanner;
public class Z16_7 {
    public static void main(String args[]) {
        System.out.println("Obliczanie silni liczby naturalnej");
        Scanner input = new Scanner(System.in);
        int n;
        do {
            System.out.print("n = ");
            n = input.nextInt();
        } while (n < 0);
        input.close();
        long silnia = 1;
        for(int i = 1; i <= n; ++i)
            silnia *= i;
        System.out.printf("%d! = %d\n", n, silnia);
    }
}
```

Nie ograniczyliśmy podczas wprowadzania danych wartości zmiennej `n`. Musimy jednak pamiętać, że wartość silni bardzo szybko rośnie i możemy obliczyć co najwyżej *20! = 2432902008176640000* (dla typu `long`). Następna wartość silni przekroczy wartość `Long.MAX_VALUE` równą *9223372036854775807*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 16.8. Z16_8.java

Obliczenia umieścimy w pętli `for` o postaci `for(;;) {instrukcja;}`. Jest to *pętla nieskończona* (nie ma inicjowania zmiennej sterującej, sprawdzania i modyfikacji warunku) i będzie wykonywana aż do chwili, gdy przerwie ją jakaś instrukcja z wnętrza bloku.

Podobny efekt można uzyskać w pętli `while(true) { instrukcja; }` lub `do { instrukcja; } while(true);`.

Wewnątrz pętli wprowadzamy dane z konsoli — podstawę `a` typu `double` i wykładnik `n` (dopuszczalną wartością jest liczba naturalna lub *–1*). Podanie wykładnika równego –1 spowoduje przerwanie pętli (`for`) i zakończenie obliczeń (`if (n == -1) break;`).

Przyjmujemy wartość początkową potęgi `double pot = 1.0;` ($a^0 = 1, \quad a \neq 0$) i następnie w pętli obliczamy kolejne potęgi ($a^1, a^2, ..., a^n$):

```
for(int i = 1; i <= n; ++i)
    pot *= a;
```

Program ma jedną wadę — oblicza wartość symbolu nieoznaczonego 0^0, podając wynik *1*. Możemy jednak taki rezultat zaakceptować (również metoda `pow()` z klasy `Math` obliczy 0^0 w ten sam sposób).

```
import java.util.Scanner;
public class Z16_8 {
    public static void main(String args[]) {
        System.out.println("Obliczanie potęg o wykładniku naturalnym");
        Scanner input = new Scanner(System.in);
        for(;;) {
            System.out.print("Podstawa, a = ");
            double a = input.nextDouble();
            int n;
            do {
                System.out.print("Wykładnik (-1 - koniec)n = ");
                n = input.nextInt();
            } while (n < -1);
            if (n == -1) break;
            double pot = 1.0;
            for(int i = 1; i <= n; ++i)
                pot *= a;
            System.out.printf("(%f)^%d = %f\n", a, n, pot);
        }
        input.close();
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 7.

Rozwiązania zadań

17. Obsługa wyjątków

Zadanie 17.1. Z17_1.java

Jeśli wczytujemy liczbę, stosując metodę `nextDouble()` (lub inną podobną) z klasy `Scanner`, i wprowadzony ciąg znaków nie jest poprawnym zapisem liczby, zostanie zgłoszony wyjątek `InputMismatchException`. Obsługując ten wyjątek, możemy zignorować błąd i powrócić do wczytywania danych.

Wczytywanie danych umieścimy w pętli ze sprawdzaniem warunku na końcu. Zakończenie lub powtórzenie pętli zależy od wartości zmiennej logicznej `end`.

```
boolean end;
do {
    /* wczytywanie danych */
} while(end);
```

Instrukcję `x = input.nextDouble()`, która może w przypadku wystąpienia błędu zgłosić wyjątek, umieszczamy w bloku `try{ }` razem z instrukcją podstawiania `end = true`. Poprawne wczytanie wartości zmiennej `x` spowoduje, że następnie zmienna `end` przyjmie wartość `true`, a w konsekwencji opuścimy pętlę wczytywania danych. Po wprowadzeniu błędnej danej (np. tekst *dwa* zamiast liczby *2*) zostanie zgłoszony wyjątek `InputMismatchException`. Obsługa tego wyjątku to wyświetlenie komunikatu, usunięcie ze strumienia `input` błędnego tokenu (`input.next()`) lub całej linii (`input.nextLine()`) i ustawienie wartości zmiennej `end = false`. Ta ostatnia czynność spowoduje powrót na początek pętli wczytywania danych.

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Z17_1 {
    public static void main(String args[]) {
        System.out.println("Wczytywanie liczb");
        Scanner input = new Scanner(System.in);
        double x = 0.0;
        boolean end;
        do {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            try {
                System.out.print("x = ");
                x = input.nextDouble();
                end = true;
            } catch(InputMismatchException e) {
                System.out.println("To nie jest liczba!");
                input.next();
                end = false;
            }
        } while(!end);
        input.close();
        System.out.printf("Wprowadzona liczba, x = %f\n", x);
    }
}
```

Możemy zrezygnować ze zmiennej logicznej `end`, tworząc nieskończoną pętlę, w której zawrzemy wczytywanie danych i obsługę wyjątku. Pętlę przerwiemy instrukcją `break` po poprawnym odczytaniu liczby ze strumienia.

```
while(true) {
    try {
        System.out.print("x = ");
            x = input.nextDouble();
            break;
    } catch(InputMismatchException e) {
        System.out.println("To nie jest liczba!");
        input.next();
    }
}
```

Zadanie 17.2. Z17_2.java

Wczytujemy łańcuch znaków ze strumienia `String s = input.next()`. To nie powoduje żadnych problemów. Konwersji łańcucha `s` na liczbę możemy dokonać, stosując metodę `x = Double.parseDouble(s)`. Metoda ta zgłosi wyjątek `NumberFormatException`, jeśli łańcuch znaków `s` nie jest poprawnym zapisem liczby. Pozostała część kodu jest podobna jak w rozwiązaniu zadania 17.1.

```
boolean end;
do {
    try {
        System.out.print("x = ");
        String s = input.next();
        x = Double.parseDouble(s);
        end = true;
    } catch(NumberFormatException e) {
        System.out.println("To nie jest liczba!");
        end = false;
    }
} while(!end);
```

Możemy również zastosować drugi wariant — przerwanie pętli instrukcją `break`.

```
for(;;) {
    try {
        System.out.print("x = ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            String s = input.next();
            x = Double.parseDouble(s);
            break;
        } catch(NumberFormatException e) {
            System.out.println("To nie jest liczba!");
        }
    }
```

Zadanie 17.3. Z17_3.java

Sposób I. Zmienna `licznik` służy do zliczania dodawanych liczb całkowitych. Jeśli wczytany token jest liczbą całkowitą, to ta liczba jest dodawana do sumy (zmienna `suma += n`) i licznik zwiększany jest o 1 (`++licznik`). Gdy token nie jest liczbą całkowitą, to rzucany jest wyjątek `NumberFormatException` i po wyświetleniu stosownego komunikatu (w bloku `catch`) powracamy do pętli wczytującej dane.

```
import java.util.Scanner;
public class Z17_3 {
    public static void main(String args[]) {
        System.out.println("Obliczanie sumy pięciu liczb całkowitych");
        System.out.println("Podaj pięć liczb całkowitych:");
        Scanner input = new Scanner(System.in);
        int suma = 0, licznik = 0;
        do {
            String s = "";
            try {
                s = input.next();
                int n = Integer.parseInt(s);
                ++licznik;
                suma += n;
                System.out.println("Liczba "+licznik+": "+n);
            } catch(NumberFormatException e) {
                System.out.println("To nie jest liczba całkowita: "+s);
            }
        } while(licznik < 5);
        input.close();
        System.out.printf("Suma liczb S = %d\n", suma);
    }
}
```

Sposób II. W tym rozwiązaniu przed wczytaniem tokenu ze skanera sprawdzimy, czy jest to liczba całkowita (metoda `hasNextInt()`). Jeśli wynik testu będzie pozytywny, to wczytamy liczbę, dodamy ją do sumy i zwiększymy licznik. W przeciwnym wypadku przeczytamy token i zignorujemy jego wartość (możemy wyświetlić odpowiedni komunikat).

```
import java.util.Scanner;
public class Z17_3a {
    public static void main(String args[]) {
        System.out.println("Obliczanie sumy pięciu liczb całkowitych");
        System.out.println("Podaj pięć liczb całkowitych:");
        Scanner input = new Scanner(System.in);
        int suma = 0, licznik = 0;
        do {
            if (input.hasNextInt()) {
                int n = input.nextInt();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                ++licznik;
                suma += n;
                System.out.println("Liczba "+licznik+": "+n);
            } else
                System.out.println("To nie jest liczba całkowita: "+input.next());
        } while(licznik < 5);
        input.close();
        System.out.printf("Suma liczb S = %d\n", suma);
    }
}
```

Zadanie 17.4. Z17_4a.java, Z17_4b.java

a) Wczytujemy łańcuch znaków `s` ze strumienia wejściowego i konwertujemy ten łańcuch na liczbę całkowitą. Jeśli `s` nie reprezentuje liczby całkowitej, to instrukcja `n = Integer.parseInt(s)` zwróci wyjątek `NumberFormatException`. Przechwycimy ten wyjątek i wyświetlimy komunikat `"To nie jest liczba całkowita: "` z dołączoną zawartością łańcucha `s`. Dzielenie zmiennoprzecinkowe `1.0/n` lub `1/(double)n` da w wyniku odwrotność liczby `n`. Dla `n` równego `0` nie zostanie zgłoszony wyjątek, ale otrzymamy wynik `Infinity` (nieskończoność). W takiej sytuacji sami zgłosimy wyjątek `throw new ArithmeticException("Nie istnieje odwrotność liczby 0.")`, a następnie go przechwycimy i obsłużymy `catch(ArithmeticException e) {System.out.println(e.getMessage());}`. Zwróćmy uwagę na użycie metody `getMessage()` do pobrania tekstu komunikatu przekazywanego przez wyjątek `e`.

```
import java.util.Scanner;
public class Z17_4a {
    public static void main(String args[]) {
        System.out.println("Odwrotność liczby całkowitej");
        Scanner input = new Scanner(System.in);
        System.out.print("n = ");
        String s = input.next();
        try {
            int n = Integer.parseInt(s);
            if (n != 0)
                System.out.println("1/n = "+1.0/n);
            else
                throw new ArithmeticException("Nie istnieje odwrotność
                                                liczby 0.");
        } catch(NumberFormatException e) {
            System.out.println("To nie jest liczba całkowita: "+s);
        }
        catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
        input.close();
    }
}
```

b) Stosując metodę `input.hasNextInt()`, sprawdzamy, czy kolejny token jest liczbą całkowitą. Jeśli tak jest, to wczytujemy wartość i przechowujemy ją w zmiennej `n` (`int n = input.nextInt()`). Gdy token nie jest liczbą, wczytujemy go jako tekst i wykorzystujemy do zbudowania odpowiedniego komunikatu.

Ebookpoint.pl kopia dla: pawher@o2.pl

Dla liczby `n` różnej od zera wykonujemy obliczenie odwrotności, dla zera wyświetlamy odpowiedni komunikat.

```
import java.util.Scanner;
public class Z17_4b {
    public static void main(String args[]) {
        System.out.println("Odwrotność liczby całkowitej");
        Scanner input = new Scanner(System.in);
        System.out.print("n = ");
        if (input.hasNextInt()) {
            int n = input.nextInt();
            if (n != 0)
                System.out.println("1/n = "+1.0/n);
            else
                System.out.println("Nie istnieje odwrotność liczby 0.");
        } else
            System.out.println("To nie jest liczba całkowita: "+input.next());
        input.close();
    }
}
```

Przewidzieliśmy i ominęliśmy, stosując instrukcje warunkowe, wszystkie sytuacje, w których mógł wystąpić błąd (bez użycia wyjątków).

Zadanie 17.5. Z17_5.java

Błąd może wystąpić podczas konwertowania wczytywanych danych (łańcuchów znaków) na liczby lub podczas dzielenia przez zero. Spowoduje to rzucenie wyjątku, który następnie przechwytujemy i obsługujemy.

```
import java.util.Scanner;
public class Z17_5 {
    public static void main(String args[]) {
        System.out.println("Dzielenie z resztą");
        Scanner input = new Scanner(System.in);
        System.out.print("m = ");
        String sm = input.next();
        System.out.print("n = ");
        String sn = input.next();
        try {
            int m = Integer.parseInt(sm);
            int n = Integer.parseInt(sn);
            System.out.printf("%d:%d = %d r. %d\n", m, n, m/n, m%n);
        } catch(NumberFormatException e) {
            System.out.println("Błąd podczas wprowadzania danych!");
        }
        catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
        input.close();
    }
}
```

Próba dzielenia przez zero spowoduje wyświetlenie komunikatu `/ by zero` (wartość `e.getMessage()` dla `ArithmeticException`). Możemy w tym miejscu wstawić własny komunikat, np. `"Dzielenie przez 0"`.

Ebookpoint.pl kopia dla: pawher@o2.pl

18. Liczby pseudolosowe i tablice jednowymiarowe — budujemy metody statyczne

Zadanie 18.1. Z18_1.java

Metoda `random` z klasy `Math` zwraca liczbę pseudolosową z przedziału $\langle 0, 1)$, po pomnożeniu przez *6* otrzymamy wartość należącą do przedziału $\langle 0, 6)$. Po rzutowaniu liczby na typ całkowity (`int`) otrzymamy jedną z wartości: *0*, *1*, *2*, *3*, *4*, *5*, a po dodaniu liczby *1* uzyskamy wynik rzutu kostką do gry, czyli wartość ze zbioru {*1*, *2*, *3*, *4*, *5*, *6*}.

Wartość wyrażenia (wynik rzutu) `1+(int)(Math.random()*6)` zwracana jest przez bezparametrową statyczną metodę (funkcję) `kostka()`.

```
public class Z18_1 {
    static int kostka(){
        return 1+(int)(Math.random()*6);
    }
    public static void main(String args[]) {
        for(int i = 0; i < 100; ++i)
            System.out.print(kostka()+" ");
    }
}
```

Tak zdefiniowana statyczna metoda `kostka()` jest dostępna w pakiecie, w którym znajduje się klasa `Z18_1`, i może być wywołana poleceniem `Z18_1.kostka()`.

Inną możliwość stwarza klasa `java.util.Random`. Najpierw tworzymy obiekt `rnd`, a następnie stosujemy metodę `rnd.nextInt(6)` zwracającą liczbę całkowitą od *0* do *5* (nieosiągającą wartości *6*). Po dodaniu liczby *1* otrzymamy wynik rzutu kostką.

```
/* Tworzymy obiekt rnd klasy Random. */
static Random rnd = new Random();
/* Tworzymy funkcję losującą rzut kostką. */
static int kostka(){
    return 1+rnd.nextInt(6);
}
```

Zadanie 18.2. Z18_2.java

Losowanie rzutów kostką zostało szczegółowo opisane w rozwiązaniu zadania 18.1. Do przechowywania wyników utworzymy statyczną, siedmioelementową tablicę liczb całkowitych `wynik`. Element o indeksie `0` możemy zastosować do przechowywania liczby rzutów (zwiększanie licznika rzutów `wynik[0] += 1`) lub pozostawić go jako niewykorzystany. Elementy o indeksach od `1` do `6` będą służyły do zliczania, ile razy wypadło *1* oczko, *2* oczka itd. Dzięki takiemu podejściu wynik rzutu kostką będzie jednocześnie indeksem elementu tablicy przechowującego liczbę rzutów, w których ten wynik wypadł (zwiększanie licznika wyników: `wynik[kostka()] += 1`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Z18_2 {
    public static int kostka(){
        return 1+(int)(Math.random()*6);
    }
    private static int[] wynik = new int[7];
    public static void main(String args[]) {
        /* Losowanie */
        for(int i = 0; i < 1000; ++i) {
            wynik[0] += 1;          // licznik rzutów
            wynik[kostka()] += 1; // licznik wyników
        }
        /* Wynik */
        System.out.println("Liczba rzutów: "+wynik[0]);
        for(int i = 1; i <= 6; ++i)
            System.out.printf("%d: %d\t - %.1f%%\n", i, wynik[i],
                              wynik[i]*100.0/wynik[0]);
    }
}
```

Uzyskane wyniki przedstawiamy, podając liczbę wystąpień każdego wyniku oraz udział procentowy (`wynik[i]*100.0/wynik[0]`). Zwróćmy uwagę na sposób obliczenia: `wynik[i]` mnożymy przez `100.0` (co daje rezultat zmiennoprzecinkowy) i następnie dzielimy przez liczbę rzutów (`wynik[0]`). Wykorzystanie w wyrażeniu liczby całkowitej `100` dałoby wynik całkowity (z pominięciem części ułamkowej procentu). Użycie w łańcuchu formatującym symbolu `%%` jest potrzebne, aby wyświetlić znak procentu (`%`).

Zadanie 18.3. Z18_3.java

Suma oczek w dwukrotnym rzucie kostką do gry przyjmuje wartości od *2* do *12*. Deklarując trzynastoelementową tablicę liczb całkowitych `wynik` do zliczania wyników, zastosujemy element `wynik[0]` do liczenia powtórzeń doświadczenia i elementy `wynik[2]`, `wynik[3]`, …, `wynik[12]` do liczenia wystąpień wszystkich możliwych wartości sumy oczek w dwóch rzutach. Element `wynik[1]` pozostanie niewykorzystany. Sposób liczenia wyników pozostaje taki sam jak w rozwiązaniu zadania 18.2. Do dodawania wyników dwóch rzutów budujemy metodę statyczną `suma`, realizującą dodawanie dwóch liczb całkowitych podanych jako parametry. Nie jest to konieczne, ale zwiększa czytelność kodu programu.

```
public class Z18_3 {
    static int kostka(){
        return 1+(int)(Math.random()*6);
    }
    static int suma(int m, int n) {
        return m+n;
    }
    private static int[] wynik = new int[13];
    public static void main(String args[]) {
        System.out.println("Suma oczek w dwukrotnym rzucie kostką");
        /* Losowanie */
        for(int i = 0; i < 3000; ++i) {
            wynik[0] += 1;                          // licznik losowań
            wynik[suma(kostka(), kostka())] += 1; // licznik wyników
        }
        /* Wynik */
        System.out.println("Liczba rzutów: "+wynik[0]);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        for(int i = 2; i < 13; ++i)
            System.out.printf("Suma %2d: %4d\t - %4.1f%%\n", i, wynik[i],
                wynik[i]*100.0/wynik[0]);
    }
}
```

Zadanie 18.4. Z18_4.java

W przedziale $\langle a, b\rangle$ znajduje się $n = 10(b-a)+1$ liczb różniących się o *0,1* (*a*; *a+0,1*; *a+0,2*, *b*). Losujemy liczbę całkowitą z zakresu od *0* do *n–1*, dzielimy tę liczbę przez *10* i dodajemy do liczby *a* — wynik znajdzie się w przedziale $\langle a, b\rangle$.

```
public class Z18_4 {
    static double rand(int a, int b) {
        int n = (b-a)*10+1;
        return a+((int)(Math.random()*n))/10.0;
    }
    public static void main(String args[]) {
        System.out.println("Losowanie liczby z przedziału <a, b>\n");
        for(int i = 0; i < 50; ++i)
            System.out.printf("%4.1f ", rand(5, 7));
    }
}
```

Zadanie 18.5. Z18_5.java

Przedział $\langle 0, 5)$ podzielimy na pięć rozłącznych przedziałów $\langle 0, 1)$, $\langle 1, 2)$, $\langle 2, 3)$, $\langle 3, 4)$ i $\langle 4, 5)$ — każdy o długości *1*. Losowanie liczb przeprowadzimy, korzystając z obiektu `rnd` klasy `Random` (`Random rnd = new Random()`). Metoda `nextDouble()` zwraca pseudolosową wartość typu `double` należącą do przedziału $\langle 0, 1)$, którą następnie mnożymy przez `5` (`x = 5*rnd.nextDouble()`) i otrzymujemy liczbę `x` z przedziału $\langle 0, 5)$. Wylosowanych liczb nie zapamiętujemy. Na bieżąco, stosując ciąg zagnieżdżonych instrukcji `if-else`, zliczamy w tablicy `n` (zadeklarowanej w klasie jako prywatna i statyczna tablica zawierająca pięć liczb całkowitych), w którym przedziale mieści się wylosowana liczba.

W zestawieniu wyników podajemy, ile razy została wylosowana liczba należąca do kolejnego przedziału (`n[i]` dla `i = 0`, `1`, `2`, `3` i `4`) oraz jaki procent wszystkich losowanych liczb zmieścił się w tym przedziale (`n[i]*100.0/500`).

```
import java.util.Random;
public class Z18_5 {
    private static int[] n = new int[5];
    public static void main(String args[]) {
        System.out.println("Losowanie liczby z przedziału <0, 5)\n");
        double x;
        Random rnd = new Random();
        for(int i = 0; i < 500; ++i) {
            x = 5*rnd.nextDouble();
            if (x < 1) n[0] +=1;
            else if (x < 2) n[1] +=1;
            else if (x < 3) n[2] +=1;
            else if (x < 4) n[3] +=1;
            else n[4] +=1;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        }
        for(int i = 0; i < 5; ++i)
            System.out.printf("<%d, %d): %4d - %4.1f%%\n", i, i+1, n[i],
                n[i]*100.0/500);
    }
}
```

Zauważając w tym konkretnym przypadku zależność pomiędzy częścią całkowitą wylosowanej liczby (`(int)x`) i numerem (indeksem) przedziału, do którego liczba `x` należy, możemy zrezygnować z instrukcji warunkowej i uprościć proces zliczania:

```
for(int i = 0; i < 500; ++i) {
    x = 5*Math.random();
    n[(int)x] += 1;
}
```

Zadanie 18.6. Z18_6.java

Do przechowywania wyników losowania użyjemy sześcioelementowej tablicy (`tmp`) liczb całkowitych typu `byte` (zakres — `128..127`). Zmienna `j` jest indeksem elementów tablicy `tmp` i wskazuje jednocześnie, ile elementów zostało już wylosowanych (wartości elementów nie mogą się powtarzać).

Po wylosowaniu i wpisaniu do tablicy `tmp` pierwszego elementu (liczba całkowita z zakresu `1..49`, rzutowana na typ `byte`) zmienna `j` przyjmuje wartość `1` (efekt postinkrementacji `j++`) — indeks następnego wolnego miejsca w tablicy `tmp`.

```
byte j = 0;
tmp[j++] = (byte)(1+49*Math.random());
```

Dalsze losowanie przebiega w pętli `do {...} while (j<6);`, która zakończy się po wylosowaniu wszystkich sześciu (różnych) wartości. Po wylosowaniu kolejnej wartości (`n`) ustawiamy zmienną logiczną `jest` na wartość `false` (zakładamy, że liczba się nie powtarza) i porównujemy wartość wylosowaną `n` z dotychczasową zawartością tablicy `tmp`.

```
for(byte i = 0; i < j; ++i)
    if (n == tmp[i]) jest = true;
```

Jeśli liczba jest w tablicy, to zmienna `jest` przyjmie wartość `true` i losowanie trzeba będzie powtórzyć. Natomiast gdy liczba `n` nie wystąpi w tablicy `tmp`, to wstawimy do tablicy tę wartość i zwiększymy `j` o `1`.

```
if (!jest)
    tmp[j++] = n;
```

Dopóki `j < 6`, powtarzamy losowanie kolejnego elementu tablicy.

Po wylosowaniu wszystkich sześciu liczb posortujemy wyniki. W tym celu użyjemy statycznej metody `Arrays.sort(wynik)`, pochodzącej z klasy `java.util.Arrays`. Na koniec wyświetlamy wyniki losowania (zastosowano pętlę typu `for each`).

```
import java.util.Arrays;
public class Z18_6 {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    static byte[] lotto() {
        byte[] tmp = new byte[6];
        byte j = 0;
        tmp[j++] = (byte)(1+49*Math.random());
        do {
            byte n = (byte)(1+49*Math.random());
            boolean jest = false;
            for(byte i = 0; i < j; ++i)
                if (n == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = n;
        } while (j < 6);
        return tmp;
    }
    public static void main(String args[]) {
        byte[] wynik = lotto();
        Arrays.sort(wynik);
        for(byte x: wynik)
            System.out.print(x+" ");
    }
}
```

Zadanie 18.7. Z18_7.java

Metoda `lotto()` została szczegółowo omówiona w rozwiązaniu zadania 18.6. W tym przykładzie wprowadzamy parametr `m` zamiast stałej wartości `6` i parametr `n` zamiast liczby `49`. Gdy użytkownik poda `n < m` (żądanie wylosowania `m` różnych wartości ze zbioru zawierającego mniej niż `m` elementów), zostanie zgłoszony wyjątek.

```
import java.util.Arrays;
public class Z18_7 {
    static byte[] lotto(byte m, byte n) {
        if (n < m)
            throw new IllegalArgumentException("Liczba elementów większa
                od liczby dopuszczalnych wartości!");
        byte[] tmp = new byte[m];
        byte j = 0;
        tmp[j++] = (byte)(1+n*Math.random());
        do {
            byte w = (byte)(1+n*Math.random());
            boolean jest = false;
            for(byte i = 0; i < j; ++i)
                if (w == tmp[i])
                    jest = true;
            if (!jest)
                tmp[j++] = w;
        } while (j < m);
        return tmp;
    }
    public static void main(String args[]) {
        byte[] wynik = lotto((byte)5, (byte)49);
        Arrays.sort(wynik);
        for(byte x: wynik)
            System.out.print(x+" ");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zamiast wypisywać wyniki w pętli, możemy, stosując statyczną metodę `toString()` z klasy `Array`, zamienić tablicę wartości na łańcuch znaków (elementy tablicy oddzielone przecinkami, zawarte w nawiasie prostokątnym).

```
System.out.println(Arrays.toString(wynik));
```

Zadanie 18.8. Z18_8.java

Utworzone metody zapiszemy w dwóch plikach: *Z18_8.java* (rozwiązanie tego zadania) i *MyRandomArray.java* — klasa, w której zgromadzimy metody opracowane w tym i kilku kolejnych zadaniach.

```
import java.util.Random;
public class MyRandomArray {
    /* Tu wpisujemy kody metod. */
}
```

Wszystkie te metody zadeklarujemy jako publiczne i statyczne.

a) Metoda `int[] rndArray(int n, int m)` zwraca tablicę zawierającą `n` liczb całkowitych wylosowanych z zakresu od `0` do `m-1`.

Do losowania utworzymy obiekt `rnd` klasy `Random` (`Random rnd = new Random()`). Wyniki losowania przechowamy w `n`-elementowej tablicy liczb całkowitych `tmp` (`int[] tmp = new int[n]`). Stosując metodę `nextInt()` z klasy `Random`, w pętli `for` losujemy i podstawiamy do tablicy pseudolosowe wartości z zakresu od `1` do `m-1` (`tmp[i] = rnd.nextInt(m)`). Na koniec zwracamy referencję do tablicy `tmp` (`return tmp`).

```
public static int[] rndArray(int n, int m) {
    Random rnd = new Random();
    int[] tmp = new int[n];
    for(int i = 0; i < n; ++i)
        tmp[i] = rnd.nextInt(m);
    return tmp;
}
```

Losowanie liczb pseudolosowych możemy również zrealizować, używając statycznej metody `random()` z klasy `Math`, bez stosowania obiektu `rnd`: `(int)(m*Math.random())`. `Math.random()` zwraca liczbę typu `double` z przedziału $\langle 0, 1)$; po pomnożeniu jej przez `m` otrzymamy liczbę należącą do przedziału $\langle 0, m)$. Rzutując wynik na typ całkowity `int`, otrzymamy jedną z wartości: `0`, `1`, `2`, … `m-1`.

b) Metoda `void arrayPrint(int[] tab)` wyświetla w konsoli elementy tablicy `tab` oddzielone pojedynczym odstępem. Według własnych upodobań można zmienić znak separujący wartości.

```
public static void arrayPrint(int[] tab) {
int n = tab.length;
for(int i = 0; i < n; ++i)
    System.out.print(tab[i]+" ");
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zwróćmy uwagę na sposób wyznaczania rozmiaru tablicy. Obiekt `tab` posiada pole `length` zawierające liczbę elementów tablicy.

c) Metoda `void arrayPrintln(int[] tab)` wyświetla w konsoli elementy tablicy `tab` oddzielone pojedynczym odstępem i przenosi kursor na początek nowego wiersza.

```
public static void arrayPrintln(int[] tab) {
    int n = tab.length;
    for(int i = 0; i < n; ++i)
        System.out.print(tab[i]+" ");
    System.out.println();
}
```

d) Metoda `void addToArray(int[] tab, int d)` do każdego elementu tablicy (podanej jako pierwszy parametr) dodaje liczbę całkowitą (drugi parametr).

```
public static void addToArray(int[] tab, int d) {
    int n = tab.length;
    for(int i = 0; i < n; ++i)
        tab[i] += d;
}
```

Działanie tych metod można zobaczyć w następującym programie:

```
import java.util.Random;
public class Z18_8 {
    /* Tu wpisujemy kody metod: rndArray(), arrayPrintln() i addToArray(). */
    public static void main(String args[]) {
        /* 10 liczb z zakresu od 0 do 99 (< 100) */
        arrayPrintln(rndArray(10, 100));
        /* 15 liczb z zakresu od 0 do 999 (< 1000) */
        int[] t = rndArray(15, 1000);
        arrayPrintln(t);
        /* 15 liczb z zakresu od 1000 do 1999 */
        addToArray(t, 1000);
        arrayPrintln(t);
    }
}
```

Zadanie 18.9. Z18_9.java

Metoda `int[] rndUniqueArray(int n, int m)` zwraca tablicę zawierającą `n` różnych liczb całkowitych wylosowanych z zakresu od `0` do `m-1`. Algorytm losowania różnych liczb został szczegółowo omówiony w rozwiązaniu zadania 18.6.

```
public static int[] rndUniqueArray(int n, int m) {
    if (m < n)
        throw new IllegalArgumentException("Liczba elementów większa
            od liczby dopuszczalnych wartości!");
    int[] tmp = new int[n];
    Random rnd = new Random();
    int j = 0;
    tmp[j++] = rnd.nextInt(m);
    do {
        int k = rnd.nextInt(m);
        boolean jest = false;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(int i = 0; i < j; ++i)
                if (k == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = k;
        } while (j < n);
        return tmp;
    }
```

Metodę tę możemy dołączyć do klasy `MyRandomArray` i wstawić do kodu źródłowego programu pokazującego działanie metody. W bieżącym folderze powinien być skompilowany plik *MyRandomArray.class* lub kod źródłowy tej klasy (korzystamy z metody `arrayPrintln()`).

```
import java.util.Random;
public class Z18_9 {
    /* Kod metody rndUniqueArray */
    public static void main(String args[]) {
        /* 10 liczb z zakresu od 0 do 99 bez powtarzania */
        MyRandomArray.arrayPrintln(rndUniqueArray(10, 100));
    }
}
```

Zadanie 18.10. Z18_10.java

1. Metoda `int[] rndSortArray(int n, int m)` zwraca posortowaną (w porządku niemalejącym) tablicę zawierającą `n` liczb całkowitych wylosowanych z zakresu od `0` do `m-1`.

   ```
   public static int[] rndSortArray(int n, int m) {
       Random rnd = new Random();
       int[] tmp = new int[n];
       for(int i = 0; i < n; ++i)
           tmp[i] = rnd.nextInt(m);
       Arrays.sort(tmp);
       return tmp;
   }
   ```

   Metoda ta działa tak samo jak metoda `int[] rndArray(int n, int m)` omówiona w rozwiązaniu zadania 18.8. Jedynym uzupełnieniem jest sortowanie tablicy `tmp` po zakończeniu losowania (`Arrays.sort(tmp)`). Do sortowania wykorzystujemy statyczną metodę `sort()` z klasy `Arrays`.

2. Metoda `int[] rndSortUniqueArray(int n, int m)` zwraca posortowaną (w porządku rosnącym) tablicę zawierającą `n` różnych liczb całkowitych wylosowanych z zakresu od `0` do `m-1`. Aby zadanie było możliwe do wykonania, liczba elementów `n` nie może być większa od liczby `m`, ograniczającej z góry zbiór wylosowanych wartości całkowitych.

   Kod metody jest kopią kodu metody `rndUniqueArray()` (przedstawionej w rozwiązaniu zadania 18.9), uzupełnioną o sortowanie tablicy po zakończeniu losowania (`Arrays.sort(tmp)`).

   Metodę dołączamy do klasy `MyRandomArray` i wstawiamy do kodu źródłowego przykładu.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.*;
public class Z18_10 {
    /* Kod metody rndSortArray */
    /* Kod metody rndSortUniqueArray */
    public static void main(String args[]) {
        /* 20 posortowanych liczb z zakresu od 0 do 9 */
        MyRandomArray.arrayPrintln(rndSortArray(20, 10));
        /* 10 posortowanych liczb z zakresu od 0 do 99 bez powtarzania */
        MyRandomArray.arrayPrintln(rndSortUniqueArray(10, 100));
    }
}
```

Zwróćmy uwagę na pierwszy wiersz przykładu `import java.util.*;`, który powoduje importowanie wszystkich klas z pakietu `java.util` (w tym klas `Arrays` i `Random` — niezbędnych do poprawnego działania tej aplikacji).

Zadanie 18.11. Z18_11.java

Na podstawie rozwiązania zadania 18.8 tworzymy następujące metody:

1. Metoda `double[] rndArray(int n, double a)` zwraca `n`-elementową tablicę liczb zmiennoprzecinkowych typu `double`, wypełnioną pseudolosowymi wartościami z przedziału $\langle 0, a)$. Do losowania wykorzystujemy obiekt `rnd` klasy `Random` i metodę `nextDouble()` zwracającą zmiennoprzecinkową liczbę typu `double` z przedziału $\langle 0, 1)$.

   ```
   public static double[] rndArray(int n, double a) {
       Random rnd = new Random();
       double[] tmp = new double[n];
       for(int i = 0; i < n; ++i)
           tmp[i] = a*rnd.nextDouble();
       return tmp;
   }
   ```

   Generowanie liczb pseudolosowych można również zrealizować, stosując statyczną metodę `random()` z klasy `Math`:

   ```
   for(int i = 0; i < n; ++i)
       tmp[i] = a*Math.random();
   ```

2. Wyświetlanie elementów tablicy możemy zrealizować przy użyciu jednej z dwóch metod, analogicznych do metod opracowanych w rozwiązaniu zadania 18.8.

   ```
   public static void arrayPrint(double[] tab) {
       int n = tab.length;
       for(int i = 0; i < n; ++i)
           System.out.print(tab[i]+" ");
   }

   public static void arrayPrintln(double[] tab) {
       int n = tab.length;
       for(int i = 0; i < n; ++i)
           System.out.print(tab[i]+" ");
       System.out.println();
   }
   ```

Ebookpoint.pl kopia dla: pawher@o2.pl

3. Metoda `void arrayPrintf(String spec, double[] tab)` wyświetla sformatowane elementy tablicy. Formatowanie określone jest z wykorzystaniem parametru `spec`, w sposób przyjęty dla metody `printf` z klasy `System`.

```
public static void arrayPrintf(String spec, double[] tab) {
    int n = tab.length;
    for(int i = 0; i < n; ++i)
        System.out.printf(spec, tab[i]);
}
```

4. Metoda `void addToArray(double[] tab, double d)` dodaje do wszystkich elementów tablicy (podanej jako pierwszy parametr) liczbę typu `double` (drugi parametr).

```
public static void addToArray(double[] tab, double d) {
    int n = tab.length;
    for(int i = 0; i < n; ++i)
        tab[i] += d;
}
```

Kody metod dołączamy do klasy `MyRandomArray` i wstawiamy do kodu źródłowego programu pokazującego ich działanie. Metody te mają nazwy identyczne z nazwami metod znajdujących się już w klasie `MyRandomArray`, ale różnią się od nich typem parametrów (występuje tu tzw. przeciążanie metod).

Działanie metod możemy przedstawić w programie:

```
import java.util.Random;
public class Z18_11 {
    /* Kod metod */
    public static void main(String args[]) {
        /* 10 liczb z przedziału <0, 5.5) */
        arrayPrintln(rndArray(10, 5.5));
        System.out.println();
        /* 5 liczb z przedziału <0, 12.5) */
        double[] t = rndArray(5, 12.5);
        arrayPrintln(t);
        System.out.println();
        /* 5 liczb z przedziału <2.5, 15.0)*/
        addToArray(t, 2.5);
        arrayPrintf("%6.1f", t);
    }
}
```

Zadanie 18.12. Z18_12.java

Stosując algorytm opisany w rozwiązaniu zadania 18.6 (losowanie niepowtarzających się elementów tablicy) i wzorując się na rozwiązaniu zadania 18.9, budujemy kod metody `rndUniqueArray()` zwracającej *n*-elementową tablicę różnych liczb typu `double` z przedziału $\langle 0, a)$.

```
public static double[] rndUniqueArray(int n, double a) {
    double[] tmp = new double[n];
    Random rnd = new Random();
    int j = 0;
    tmp[j++] = a*rnd.nextDouble();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        do {
            double x = a*rnd.nextDouble();
            boolean jest = false;
            for(int i = 0; i < j; ++i)
                if (x == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = x;
        } while (j < n);
        return tmp;
    }
```

Dodajemy metodę do klasy `MyRandomArray` i do kodu przykładu:

```
import java.util.Random;
public class Z18_12 {
    /* Kod metody rndUniqueArray */
    public static void main(String args[]) {
        /* 10 liczb z przedziału <0, 7.5) bez powtarzania wartości */
        MyRandomArray.arrayPrintln(rndUniqueArray(10, 7.5));
    }
}
```

Zadanie 18.13. Z18_13.java

Kody metod `double[] rndSortArray(int n, double a)` i `double[] rndSortUniqueArray(int n, double a)` uzyskamy, kopiując kody metod `double[] rndArray(int n, double a)` i `double[] rndUniqueArray(int n, double a)` (zob. rozwiązania zadań 18.11 i 18.12) i wstawiając sortowanie tablicy `tmp` po losowaniu liczb (`Arrays.sort(tmp)`). Obie metody dołączmy do klasy `MyRandomArray`.

```
import java.util.Random;
import java.util.Arrays;
public class Z18_13 {
    public static double[] rndSortArray(int n, double a) {
        Random rnd = new Random();
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = a*rnd.nextDouble();
        Arrays.sort(tmp);
        return tmp;
    }
    public static double[] rndSortUniqueArray(int n, double a) {
        double[] tmp = new double[n];
        Random rnd = new Random();
        int j = 0;
        tmp[j++] = a*rnd.nextDouble();
        do {
            double x = a*rnd.nextDouble();
            boolean jest = false;
            for(int i = 0; i < j; ++i)
                if (x == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = x;
        } while (j < n);
        Arrays.sort(tmp);
        return tmp;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    }
    public static void main(String args[]) {
        /* 20 posortowanych liczb z przedziału <0, 2.5) */
        MyRandomArray.arrayPrintln(rndSortArray(20, 2.5));
        /* 10 posortowanych liczb z przedziału <0, 10) bez powtarzania */
        MyRandomArray.arrayPrintf("%10.3f", rndSortUniqueArray(10, 10.0));
    }
}
```

Zadanie 18.14. Z18_14.java

Przedstawmy algorytm zaokrąglania ułamków dziesiętnych na przykładzie dwóch ułamków ⅓ i ⅔ zamienionych na ułamki dziesiętne. Zaokrąglenia dokonamy do 3 miejsc po przecinku.

Ułamek zwykły	*⅓*	*⅔*
Ułamek dziesiętny (`x`)	*0,333333333333...*	*0,666666666666...*
Liczba miejsc po przecinku (`prec`)	*3*	*3*
Współczynnik (`wsp`)	$10^3 = 1000$	$10^3 = 1000$
Wyrażenie: `x*wsp`	*333,333333333...*	*666,666666666...*
Wyrażenie: `x*wsp+0.5`	*333,833333333...*	*667,166666666...*
Wyrażenie: `(int)(x*wsp+0.5)`	*333*	*667*
Wyrażenie: `(int)(x*wsp+0.5)/wsp`	*0,333*	*0,667*

Otrzymany wynik jest zgodny z regułami zaokrągleń ułamków dziesiętnych. Przedstawione obliczenia wykonamy w pętli dla wszystkich elementów tablicy.

```
public class Z18_14 {
    public static void roundArray(double[] tab, int prec) {
        int n = tab.length;
        double wsp = Math.pow(10.0, prec);
        for(int i = 0; i < n; ++i)
            tab[i] = (int)(tab[i]*wsp+0.5)/wsp;
    }

    public static void main(String args[]) {
        /* 20 posortowanych liczb z przedziału <0, 2.5) */
        double[] t = MyRandomArray.rndSortArray(20, 2.5);
        roundArray(t, 2);
        MyRandomArray.arrayPrintln(t);
    }
}
```

Możemy zbudować statyczną metodę `round()`, zaokrąglającą liczbę zmiennoprzecinkową z określoną precyzją dla liczb typu `double`:

```
public static double round(double x, int prec) {
    double wsp = Math.pow(10.0, prec);
    return(int)(x*wsp+0.5)/wsp;
}
```

lub dla liczb typu `float`:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public static float round(float x, int prec) {
    float wsp = (float) Math.pow(10.0, prec);
    return (int)(x*wsp+0.5)/wsp;
}
```

Metody dodajemy do klasy `MyRandomArray`.

Zadanie 18.15. Z18_15.java

Losowane liczby mają precyzję odpowiadającą precyzji liczb typu `double`. Metoda `double[] rndArray(int n, double a, int prec)` zwraca `n`-elementową tablicę liczb zmiennoprzecinkowych, zaokrąglonych do określonej parametrem `prec` ilości miejsc po przecinku.

```
public static double[] rndArray(int n, double a, int prec) {
    Random rnd = new Random();
    double wsp = Math.pow(10.0, prec);
    double[] tmp = new double[n];
    for(int i = 0; i < n; ++i)
        tmp[i] = (int)(a*rnd.nextDouble()*wsp+0.5)/wsp;
    return tmp;
}
```

Jeśli zdefiniowaliśmy w klasie `MyRandomArray` metodę `round()`, to możemy teraz z niej skorzystać:

```
public static double[] rndArray(int n, double a, int prec) {
    Random rnd = new Random();
    double[] tmp = new double[n];
    for(int i = 0; i < n; ++i)
        tmp[i] = MyRandomArray.round(a*rnd.nextDouble(), prec);
    return tmp;
}
```

Inną możliwość stwarza zdefiniowanie metody losującej liczbę z danego zakresu i zaokrągloną z określoną precyzją, np.:

```
public static double random(double a, int prec) {
    return MyRandomArray.round(a*Math.random(), prec);
}
```

i wykorzystanie tej metody do tworzenia tablicy:

```
public static double[] rndArray(int n, double a, int prec) {
    double[] tmp = new double[n];
    for(int i = 0; i < n; ++i)
        tmp[i] = MyRandomArray.random(a, prec);
    return tmp;
}
```

Działanie metody pokazuje przykład:

```
import java.util.Random;
public class Z18_15 {
    /* Kod metody rndArray(int n, double a, int prec) */
    public static void main(String args[]) {
        /* 10 liczb z przedziału <0, 5.5) z precyzją do 2 miejsc po przecinku */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        MyRandomArray.arrayPrintln(rndArray(10, 5.5, 2));
        System.out.println();
        /* 5 liczb z przedziału <0, 12.5) z precyzją do 3 miejsc po przecinku */
        double[] t = rndArray(5, 12.5, 3);
        MyRandomArray.arrayPrintln(t);
    }
}
```

Zadanie 18.16. Z18_16.java

Metoda `rndArray()` wywołana z parametrami `5, 1.7, 0.2` powinna zwrócić pięcioelementową tablicę liczb należących do przedziału $\langle 0;1{,}7)$, różniących się o wielokrotność liczby *0,2*, czyli należących do zbioru $\{0{,}0;\ 0{,}2;\ 0{,}4;\ 0{,}6;\ 0{,}8;\ 1{,}0;\ 1{,}2;\ 1{,}4;\ 1{,}6\}$. Ponieważ w zapisie binarnym liczba *0,2* jest ułamkiem nieskończonym okresowym, to w wyniku dodawania lub mnożenia przez liczbę całkowitą na dalszych miejscach po przecinku pojawią się cyfry różne od *0*. Aby tę niedogodność wyeliminować, zastosujemy zaokrąglenie wyniku do określonej liczby miejsc po przecinku.

Najpierw obliczymy ilość liczb spełniających warunek zadania `int k = (int)(a/step)+1` (rozmieszczonych w przedziale $\langle 0, a)$, zaczynając od `0`, z krokiem określonym parametrem `step`). Precyzję zaokrąglenia (liczbę miejsc po przecinku) oszacujemy na podstawie wzoru `1+(int)Math.log10(1/step)`. Wartość elementu tablicy wyznaczymy, mnożąc krok (`step`) przez wylosowaną liczbę całkowitą z zakresu od `0` do `k-1`.

```
import java.util.Random;
public class Z18_16 {
    public static double[] rndArray(int n, double a, double step) {
        Random rnd = new Random();
        int k = (int)(a/step)+1;
        double wsp = Math.pow(10.0, 1+(int)Math.log10(1/step));
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (int)(step*rnd.nextInt(k)*wsp+0.5)/wsp;
        return tmp;
    }
    public static void main(String args[]) {
        /* 20 liczb z przedziału <0, 5.5) z krokiem 0.2 */
        MyRandomArray.arrayPrintln(rndArray(20, 5.5, 0.2));
        System.out.println();
        /* 50 liczb z przedziału <0, 12.5) z krokiem 0.05 */
        double[] t = rndArray(50, 12.5, 0.05);
        MyRandomArray.arrayPrintln(t);
    }
}
```

Metodę `double[] rndArray(int n, double a, double step)` dołączamy do klasy `MyRandomArray`.

Zadanie 18.17. Z18_17.java

Najpierw wyznaczymy rozmiar tablicy przekazanej do metody `inputArray()` (parametr `int[] tab` lub `double[] tab`). W tym celu wykorzystamy pole `length` obiektu

Ebookpoint.pl kopia dla: pawher@o2.pl

tab (int n = tab.length). Wczytywanie danych, przy użyciu obiektu input klasy Scanner, zorganizujemy w pętli for. W ten sposób możemy zbudować metody wczytywania danych do tablic z elementami dowolnego typu.

```
import java.util.Scanner;
public class Z18_17 {
    public static void inputArray(int[] tab) {
        int n = tab.length;
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; ++i) {
            System.out.printf("Liczba %d z %d, n = ", i+1, n);
            tab[i] = input.nextInt();
        }
    }
    public static void inputArray(double[] tab) {
        int n = tab.length;
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; ++i) {
            System.out.printf("Liczba %d z %d, x = ", i+1, n);
            tab[i] = input.nextDouble();
        }
    }

    public static void main(String args[]) {
        int[] a = new int[5];
        inputArray(a);
        MyRandomArray.arrayPrintln(a);
        double[] b = new double[3];
        inputArray(b);
        MyRandomArray.arrayPrintln(b);
    }
}
```

Zdefiniowane metody umieścimy w klasie MyArray.

Zadanie 18.18. Z18_18.java

Mając tablicę tab zawierającą dane jednego typu, tworzymy tablicę tmp o takiej samej liczbie elementów (int n = tab.length). Stosując rzutowanie danych liczbowych z jednego typu na drugi typ, przepisujemy w pętli zawartość tablicy tab do tablicy tmp.

```
public class Z18_18 {
    public static double[] intArrayToDouble(int[] tab) {
        int n = tab.length;
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (double)tab[i];
        return tmp;
    }

    public static int[] doubleArrayToInt(double[] tab) {
        int n = tab.length;
        int[] tmp = new int[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (int)tab[i];
        return tmp;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    }

    public static void main(String args[]) {
        double[] t = MyRandomArray.rndArray(10, 6, 3);
        MyArray.arrayPrintln(t);
        int[] u = doubleArrayToInt(t);
        MyArray.arrayPrintln(u);
        t = intArrayToDouble(u);
        MyArray.arrayPrintln(t);
    }
}
```

Zdefiniowane metody umieścimy w klasie `MyArray`. W ten sposób możemy utworzyć inne metody konwertujące tablice.

19. Dokumentacja klasy

Zadanie 19.1.

Do tworzenia dokumentacji klas używamy aplikacji *javadoc.exe* dostępnej w JDK. W plikach źródłowych musimy umieścić specjalne komentarze dokumentacyjne zaczynające się od znaku `/**` i zakończone znakiem `*/`.

Aplikację uruchamiamy z opcjonalnymi parametrami:

```
javadoc [options] [packagenames] [sourcefiles] [@files]
```

Informacje o działaniu aplikacji uzyskamy, wpisując polecenie `javadoc -help`. Również po uruchomieniu aplikacji bez parametrów otrzymamy informację o jednym błędzie, a w konsoli pojawi się poniższy tekst (tłumaczenie tekstu pozostawiamy Czytelnikowi) zawierający dostępne opcje aplikacji:

- `-overview <file>` — *Read overview documentation from HTML file*,
- `-public` — *Show only public classes and members*,
- `-protected` — *Show protected/public classes and members (default)*,
- `-package` — *Show package/protected/public classes and members*,
- `-private` — *Show all classes and members*,
- `-help` — *Display command line options and exit*,
- `-doclet <class>` — *Generate output via alternate doclet*,
- `-docletpath <path>` — *Specify where to find doclet class files*,
- `-sourcepath <pathlist>` — *Specify where to find source files*,
- `-classpath <pathlist>` — *Specify where to find user class files*,
- `-exclude <pkglist>` — *Specify a list of packages to exclude*,

Ebookpoint.pl kopia dla: pawher@o2.pl

- `-subpackages <subpkglist>` — *Specify subpackages to recursively load*,
- `-breakiterator` — *Compute 1st sentence with BreakIterator*,
- `-bootclasspath <pathlist>` — *Override location of class files loaded by the bootstrap class loader*,
- `-source <release>` — *Provide source compatibility with specified release*,
- `-extdirs <dirlist>` — *Override location of installed extensions*,
- `-verbose` — *Output messages about what Javadoc is doing*,
- `-locale <name>` — *Locale to be used, e.g.* `en_US` *or* `en_US_WIN`,
- `-encoding <name>` — *Source file encoding name*,
- `-quiet` — *Do not display status messages*,
- `-J<flag>` — *pass* `<flag>` *Directly to the runtime system*.

Opcje dostępne pod warunkiem stosowania standardowego *docletu*:

- `-d <directory>` — *Destination directory for output files*,
- `-use` — *Create class and package usage pages*,
- `-version` — *Include* `@version` *paragraphs*,
- `-author` — *Include* `@author` *paragraphs*,
- `-docfilessubdirs` — *Recursively copy doc-file subdirectories*,
- `-splitindex` — *Split index into one file per letter*,
- `-windowtitle <text>` — *Browser window title for the documentation*,
- `-doctitle <html-code>` — *Include title for the overview page*,
- `-header <html-code>` — *Include header text for each page*,
- `-footer <html-code>` — *Include footer text for each page*,
- `-top <html-code>` — *Include top text for each page*,
- `-bottom <html-code>` — *Include bottom text for each page*,
- `-link <url>` — *Create links to javadoc output at* `<url>`,
- `-linkoffline <url> <url2>` — *Link to docs at* `<url>` *using package list at* `<url2>`,
- `-excludedocfilessubdir <name1>:...` — *Exclude any doc-files subdirectories with given name*,
- `-group <name> <p1>:<p2>:...` — *Group specified packages together in overview page*,
- `-nocomment` — *Supress description and tags, generate only declarations*,
- `-nodeprecated` — *Do not include* `@deprecated` *information*,

Ebookpoint.pl kopia dla: pawher@o2.pl

- `-noqualifier <name1>:<name2>:...` — *Exclude the list of qualifiers from the output*,
- `-nosince` — *Do not include* `@since` *information*,
- `-notimestamp` — *Do not include hidden time stamp*,
- `-nodeprecatedlist` — *Do not generate deprecated list*,
- `-notree` — *Do not generate class hierarchy*,
- `-noindex` — *Do not generate index*,
- `-nohelp` — *Do not generate help link*,
- `-nonavbar` — *Do not generate navigation bar*,
- `-serialwarn` — *Generate warning about* `@serial` *tag*,
- `-tag <name>:<locations>:<header>` — *Specify single argument custom tags*,
- `-taglet` — *The fully qualified name of Taglet to register*,
- `-tagletpath` — *The path to Taglets*,
- `-charset <charset>` — *Charset for cross-platform viewing of generated documentation*,
- `-helpfile <file>` — *Include file that help link links to*,
- `-linksource` — *Generate source in HTML*,
- `-sourcetab <tab length>` — *Specify the number of spaces each tab takes up in the source*,
- `-keywords` — *Include HTML meta tags with package, class and member info*,
- `-stylesheetfile <path>` — *File to change style of the generated documentation*,
- `-docencoding <name>` — *Output encoding name*.

Zadanie 19.2. Z19_2.java

Postępując zgodnie z treścią zadania i załączoną uwagą, uruchomimy w konsoli polecenie:

```
javadoc -d Z19_2 Z19_2.java
```

i zobaczymy raport z przebiegu tworzenia dokumentacji:

```
Loading source file Z19_2.java...
Constructing Javadoc information...
Standard Doclet version 1.6.0_27

Building tree for all the packages and classes...
Generating Z19_2\Z19_2.html...
Generating Z19_2\package-frame.html...
Generating Z19_2\package-summary.html...
Generating Z19_2\package-tree.html...
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
Generating Z19_2\constant-values.html...

Building index for all the packages and classes...
Generating Z19_2\overview-tree.html...
Generating Z19_2\index-all.html...
Generating Z19_2\deprecated-list.html...

Building index for all classes...
Generating Z19_2\allclasses-frame.html...
Generating Z19_2\allclasses-noframe.html...
Generating Z19_2\index.html...
Generating Z19_2\help-doc.html...
Generating Z19_2\stylesheet.css...
```

W folderze *Z19_2* utworzono 12 plików HTML, kaskadowy arkusz stylów *stylesheet.css*, plik *package-list* (w tym przypadku pusty) oraz folder *resources* zawierający obraz *inherit.gif* (∟). Dla nas najważniejszy będzie plik *index.html* (główny plik dokumentacji) oraz plik *Z19_2.html* zawierający opis klasy `Z19_2`.

Kod źródłowy nie zawierał żadnych komentarzy, zatem w pliku *Z19_2.html* nie znajdziemy zbyt wielu informacji. Możemy jednak dowiedzieć się, że:

- Klasa `Z19_2` dziedziczy po klasie `java.lang.Object`.
- Kompilator utworzył domyślny konstruktor `Z19_2()`, co pozwala tworzyć obiekty naszej klasy, np. `Z19_2 x = new Z19_2()`.
- Klasa posiada publiczną i statyczną metodę `main` (nagłówek: `public static void main(java.lang.String[] args)`), czyli jest aplikacją uruchamianą poleceniem `java Z19_2` w konsoli (po uprzednim skompilowaniu pliku źródłowego na plik *Z19_2.class*).

Nie znajdziemy natomiast w dokumentacji informacji o:

- `private static final String HELLO = "Hello World!";` — prywatnej statycznej stałej `HELLO` (a właściwie zmiennej, której wartości nie można zmienić — to gwarantuje użyte słowo kluczowe `final`), zawierającej łańcuch znaków `Hello World!`
- `static String hello() { return HELLO; }` — statycznej metodzie (widocznej w pakiecie domyślnym, czyli dla wszystkich klas, których kody znajdują się w tym samym folderze) `hello()`.

a) Dokumentację tworzymy poleceniem `javadoc -d Z19_2a -nohelp Z19_2.java`. Nie został utworzony plik *help-doc.html* (opis zasad tworzenia dokumentacji) i nie ma linków do tego pliku.

b) Dokumentację tworzymy poleceniem `javadoc -d Z19_2b -notree Z19_2.java`. Dokumentacja nie zawiera plików *package-tree.html* i *overview-tree.html* (nie ma hierarchii klas przedstawionej w postaci drzewa).

c) Dokumentację tworzymy poleceniem `javadoc -d Z19_2c -noindex Z19_2.java`. Tym razem nie utworzono pliku *index-all.html* zawierającego alfabetyczny wykaz metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

d) Dokumentację tworzymy poleceniem `javadoc -d Z19_2d -nodeprecatedlist Z19_2.java`. Pomijamy w ten sposób plik *deprecated-list.html* zawierający listę *zdeprecjonowanych* elementów języka (niezalecanych ze względu na obniżoną jakość i możliwych do zastąpienia nowszymi elementami).

e) Dokumentację tworzymy poleceniem `javadoc -d Z19_2e -nodeprecatedlist -noindex -notree -nohelp -private Z19_2.java`, eliminując elementy omówione w punktach a), b), c) i d) oraz włączając do dokumentacji wszystkie elementy klas (również prywatne). Tym razem w dokumentacji pojawiła się informacja o prywatnych elementach widocznych w klasie i pakiecie (domyślnym):

- `private static final java.lang.String HELLO` — prywatna (w zakresie klasy), stała wartość pola (ang. *Constant Field Values*), zawierająca łańcuch `"Hello World!"`.
- `(package private) static java.lang.String hello()` — prywatna (w zakresie pakietu), statyczna metoda `hello()`, zwracająca łańcuch znaków.

W przedstawiony sposób Czytelnik może samodzielnie przetestować działanie innych opcji.


Uwaga

Czytelnik może zobaczyć taki komunikat:

```
javadoc: error - Illegal package name: "-d"
Loading source file Z19_2.java...
Loading source files for package Z19_2...
javadoc: warning - No source files for package Z19_2
1 error
1 warning
```

Dzieje się tak, jeśli w ścieżce do programu *javadoc* jest spacja (a prawdopodobnie będzie, bo standardowo środowisko Javy jest zainstalowane w *Program Files*). Podanie pełnej ścieżki dostępu do *javadoc* (w cudzysłowie) rozwiąże ten problem.

Zadanie 19.3. Z19_3.java

Komentarz dokumentacyjny zaczyna się od symbolu `/**` i jest zakończony symbolem `*/`. Znaki * rozpoczynające każdy wiersz mają wyłącznie charakter ozdobników i nie wchodzą w skład tekstu dokumentacji. Są ignorowane podczas tworzenia dokumentacji. Również łamanie wierszy w edytorze tekstu jest ignorowane.

Pierwszy komentarz zawiera ogólne informacje o klasie `Z19_3`. Zwróćmy uwagę na użyte w komentarzu znaczniki języka HTML:

- `<code></code>` — fragmenty kodu prezentowane w dokumentacji, nazwy metod, zmiennych itp.,
- `<b></b>` — wyróżnienie tekstu pismem pogrubionym (*bold*),
- `<i></i>` — wyróżnienie tekstu pismem pochylonym (*italic*),
- `<p></p>` — akapit (*paragraph*),
- `<a href="..."></a>` — link do innego dokumentu.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Klasa <code>Z19_3</code> stanowi prosty przykład aplikacji
 * konsolowej. Początki programowania w wielu językach zawierają ten
 * klasyczny przykład - wyświetlanie w konsoli napisu <b>Hello
 * World!</b>, czyli <i>Witaj, świecie!</i>.
 * <p>To radosny okrzyk, wydawany przez każdego początkującego
 * programistę po napisaniu pierwszego prostego programu w poznawanym
 * języku.</p>
 * <p>Zobacz przykłady takich programów:
 * <a href="http://en.wikipedia.org/wiki/Hello_world_program_examples"
 * target="_blank"> Hello world program examples.</a></p>
 */
public class Z19_3 {
```

Drugi komentarz odnosi się do stałej `HELLO`. Do łamania wierszy użyto znacznika `<br>`, wcięcie w kodzie uzyskano, stosując znak spacji nierozdzielającej ` `. Ten komentarz pojawi się w dokumentacji, gdy podczas jej tworzenia użyjemy opcji `-private`.

```
/** Identyfikator stałej (zmiennej, której wartości nie możemy zmienić),
 * zawierającej łańcuch znaków <code>Hello World!"</code>. Zwyczajowo
 * takie identyfikatory zapisujemy wielkimi literami. Prywatna stała
 * <code>HELLO</code> jest dostępna wyłącznie wewnątrz klasy
 * <code>Z19_3</code>. Zastosowanie stałej przedstawimy w dwóch
 * przykładach:
 * <p><b>Przykład 1.</b>
 * <code><br>static String hello() {<br>&amp;nbsp;&amp;nbsp;&amp;nbsp;&amp;nbsp;
 * return HELLO;<br>}</code></p>
 * <p><b>Przykład 2.</b>
 * <code><br>System.out.println("Jeszcze raz: "+HELLO");</code></p>
 */
   private static final String HELLO = "Hello World!";
```

Kolejny komentarz przedstawia metodę `hello()`:

```
/** Metoda zwraca łańcuch znaków - jest to napis zawarty w stałej
 * <code>HELLO</code>. Metoda jest widoczna w pakiecie. Metody z innych
 * klas pakietu mogą tę metodę wywołać: <code>Z19_3.hello();</code>
 */
   static String hello() {
       return HELLO;
   }
```

Ostatni komentarz przekazuje informacje o metodzie `main()` i możliwości jej wykorzystania. Metoda `main()` posiada jeden parametr `args` będący tablicą parametrów (w postaci łańcuchów znaków) przekazywaną przez system operacyjny podczas uruchamiania aplikacji. Zwróćmy uwagę na sposób udokumentowania tego parametru (`@param identyfikator_argumentu opis_argumentu...`).

```
/** Metoda <code>main()</code> jest miejscem, od którego rozpoczyna się
 * wykonywanie programu. Program może składać się z wielu klas, ale
 * tylko jedna z nich może zawierać publiczną, statyczną i niezwracającą
 * żadnej wartości (<code>public static void</code>) metodę
 * <code>main()</code>. Metoda <code>main()</code>
 * posiada obowiązkowo jeden parametr.<br>
 * Metodę <code>main()</code> możemy uruchomić wewnątrz innej klasy:
 * <p><b>Przykład 1.</b><code><br>Z19_3.main(null);<br>}<br></code>
 * Do metody <code>main()</code> z klasy <code>Z19_3</code> nie
 * przekazano żadnych parametrów.</p>
 * <p><b>Przykład 2.</b><code><br>String[] p = {"par1", "par2"};<br>
 * Z19_3.main(p);<br></code> Do metody <code>main()</code> z klasy
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
 * <code>Z19_3</code> przekazano dwa parametry <code>"par1"</code>
 * i <code>"par2".</code></p>
 * @param args tablica argumentów (egzemplarzy klasy <code>String
 * </code>) przekazywana przez system do metody <code>main()</code>.
 * Identyfikator <code>args</code> jest skrótem słowa "argumenty" (ang.
 * <i>arguments</i>) i może być zastąpiony dowolnym innym, poprawnie
 * zbudowanym identyfikatorem.
 */
    public static void main(String[] args) {
        System.out.println(hello());
        System.out.println("Jeszcze raz: "+HELLO);
    }
}
```

Należy też zwrócić uwagę na pierwsze zdanie w komentarzu. To zdanie (do pierwszej kropki) pojawia się w opisie skróconym stałej, zmiennej lub metody (`Summary`). Cały tekst natomiast pojawi się w opisie szczegółowym (`Detail`).

Dokumentacja została utworzona poleceniem `javadoc -d Z19_3 -nodeprecatedlist -nohelp -private Z19_3.java` i znajduje się w folderze *Z19_3*.

Podczas kompilacji automatycznie tworzony jest domyślny, bezparametrowy konstruktor klasy (pod warunkiem, że użytkownik nie zdefiniował własnej wersji bezparametrowego konstruktora). Jeśli chcemy opisać bezparametrowy konstruktor `Z19_3()`, to musimy podać jego definicję poprzedzoną odpowiednim komentarzem dokumentacyjnym. Oto przykład:

```
/** Bezparametrowy konstruktor <code>Z19_3()</code> pozwala na tworzenie
 * obiektów klasy <code>Z19_3</code> przez inne klasy. Umożliwia to
 * wywoływanie metody <code>hello()</code> i metody <code>main()</code>
 * zdefiniowanej w klasie <code>Z19_3</code> przez te klasy.<br>
 * W przypadku metody <code>main()</code> musimy przekazać parametr -
 * tablicę argumentów. Parametr ten może być tablicą pustą.<br>
 * <p><b>Przykład</b><br>
 * <code>Z19_3 x = new Z19_3();<br>
 * System.out.println(x.hello());<br>
 * x.main(null);<br></code></p>
 */
public Z19_3(){}
```


Uwaga

Jeśli w uzyskanej dokumentacji pojawią się problemy z wyświetlaniem liter z polskimi znakami diakrytycznymi, należy zapisać pliki źródłowe z odpowiednim kodowaniem i zastosować przełącznik `-charset` przy generowaniu dokumentacji.

Zadanie 19.4. MyIntArray.java

W komentarzu do klasy umieszczamy dwa znaczniki informujące o autorze i wersji programu:

```
@author Wiesław Rychlicki
@version 1.0 (2012-02-06)
```

Aby informacje przekazane za pomocą tych znaczników ukazały się w dokumentacji, należy tworzyć ją z opcjami `-author -version`. W przykładowym rozwiązaniu zadania zmieniono (skrócono) nazwy metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
/** Klasa <code>MyIntArray</code> zawiera statyczne metody, ułatwiające
 * wprowadzanie danych do jednowymiarowej tablicy liczb całkowitych,
 * wyświetlanie zawartości tablicy w konsoli oraz konwertowanie tablicy
 * liczb zmiennoprzecinkowych na tablicę liczb całkowitych.
 * @author Wiesław Rychlicki
 * @version 1.0 (2012-02-06)
 */
public class MyIntArray {
/** Umożliwia wprowadzanie danych liczbowych (typu <code>int</code>)
 * z konsoli do tablicy. Dane wprowadzane są przy użyciu metod z klasy
 * <code>java.util.Scanner</code>. Podczas wprowadzania danych
 * wyświetlana jest informacja w postaci <code>Liczba 2 z 5, n = </code>.
 * @param tab jednowymiarowa tablica liczb całkowitych.
 */
    public static void input(int[] tab) {
        int n = tab.length;
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; ++i) {
            System.out.printf("Liczba %d z %d, n = ", i+1, n);
            tab[i] = input.nextInt();
        }
    }
/** Umożliwia wyświetlanie liczb zawartych w tablicy w konsoli.
 * Kolejne liczby oddzielane są odstępem.
 * @param tab jednowymiarowa tablica liczb całkowitych.
 */
    public static void print(int[] tab) {
        int n = tab.length;
        for(int i = 0; i < n; ++i)
            System.out.print(tab[i]+" ");
    }
/** Umożliwia wyświetlanie liczb zawartych w tablicy w konsoli.
 * Kolejne liczby oddzielane są odstępem. Po wyświetleniu ostatniej
 * liczby przesyłany jest do konsoli znak końca linii.
 * @param tab jednowymiarowa tablica liczb całkowitych.
 */
    public static void println(int[] tab) {
        int n = tab.length;
        for(int i = 0; i < n; ++i)
            System.out.print(tab[i]+" ");
        System.out.println();
    }
/** Zwraca jednowymiarową tablicę liczb typu <code>double</code>
 * o wartościach równych wartościom tablicy liczb całkowitych
 * przekazanej jako parametr.
 * @param tab jednowymiarowa tablica liczb całkowitych.
 */
     public static double[] toDoubleArray(int[] tab) {
        int n = tab.length;
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (double)tab[i];
        return tmp;
    }
/** Zwraca jednowymiarową tablicę liczb typu <code>int</code>,
 * o wartościach odpowiadających (zaokrąglenie w dół) wartościom
 * elementów tablicy liczb zmiennoprzecinkowych typu <code>double</code>
 * przekazanej jako parametr.
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    * @param tab jednowymiarowa tablica liczb zmiennoprzecinkowych typu
    * <code>double</code>.
    */
       public static int[] toIntArray(double[] tab) {
           int n = tab.length;
           int[] tmp = new int[n];
           for(int i = 0; i < n; ++i)
               tmp[i] = (int)tab[i];
           return tmp;
       }
}
```

Dokumentację utworzono poleceniem `javadoc -d MyIntArray -author -version -nodeprecatedlist -nohelp -private MyIntArray.java`.

Zadanie 19.5. MyDoubleArray.java

Kody metod i komentarze dokumentacyjne różnią się od przedstawionych w rozwiązaniu zadania 19.4 jedynie użytymi typami zmiennych. Dodano dwie dodatkowe metody. Metoda `printf()` służy do wyświetlania liczb sformatowanych z określoną ilością miejsc po przecinku.

```
/** Umożliwia wyświetlanie sformatowanych liczb (zapisanych w tablicy) na
  * konsoli. Sposób formatowania określony jest z wykorzystaniem parametru
  * <code>spec</code>, w sposób określony w klasie
  * <a href="http://docs.oracle.com/javase/1.5.0/docs/api/java/util/
Formatter.html">
  * Formatter</a>, np.: <code>"%.2f "</code> - wyświetlanie liczb
  * zmiennoprzecinkowych z dwoma miejscami po przecinku.
  * @param spec łańcuch znaków określający format liczby.
  * @param tab jednowymiarowa tablica liczb typu <code>double</code>.
  */
     public static void printf(String spec, double[] tab) {
         int n = tab.length;
         for(int i = 0; i < n; ++i)
             System.out.printf(spec, tab[i]);
     }
```

Ponadto oprócz metody konwertującej tablicę liczb całkowitych tupu `int` na tablicę typu `double` (metoda `toDoubleArray()`) zdefiniowano metodę konwertującą tablicę liczb typu `float` na tablicę liczb typu `double`. Stosujemy przeciążenie metody, czyli obie metody mają tę samą nazwę i typ wyniku, a różnią się typem parametru.

```
/** Zwraca jednowymiarową tablicę liczbę typu <code>double</code>
  * o wartościach równych wartościom elementów tablicy liczb
  * zmiennoprzecinkowych typu <code>float</code> przekazanej jako
  * parametr.
  * @param tab jednowymiarowa tablica liczb zmiennoprzecinkowych typu
  * <code>float</code>.
  */
     public static double[] toDoubleArray(float[] tab) {
         int n = tab.length;
         double[] tmp = new double[n];
         for(int i = 0; i < n; ++i)
             tmp[i] = (double)tab[i];
         return tmp;
     }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Dokumentację utworzono poleceniem `javadoc -d MyDoubleArray -author -version -nodeprecatedlist -nohelp -private MyDoubleArray.java`.

Zadanie 19.6. MyFloatArray.java

Wystarczy zapisać plik *MyDoubleArray.java* pod nazwą *MyFloatArray.java* i dokonać poprawek w typach danych, zastępując typ `double` typem `float` (i na odwrót w jednej z metod). Dokumentację klasy `MyFloatArray` tworzymy poleceniem `javadoc -d MyFloatArray -author -version -nodeprecatedlist -nohelp -private MyFloatArray.java`.

Wspólną dokumentację trzech klas utworzymy poleceniem `javadoc -d MyArrays -author -version -nodeprecatedlist -nohelp -private MyIntArray.java MyDoubleArray.java MyFloatArray.java`.

Zadanie 19.7. MyRandomArray.java

Dokumentację utworzymy poleceniem `javadoc -d MyRandomArray -author -version -nodeprecatedlist -nohelp -private MyRandomArray.java`. Kompletny kod klasy `MyRandomArray` wraz z pełnym komentarzem zamieszczamy na listingu. Analizę kodu i komentarzy oraz konfrontację tych informacji z utworzoną dokumentacją (HTML) pozostawiamy Czytelnikowi.

```
import java.util.Random;
import java.util.Arrays;

/** Klasa <code>MyRandomArray</code> zawiera statyczne metody
 * wypełniające jednowymiarowe tablice liczb całkowitych lub
 * zmiennoprzecinkowych wartościami pseudolosowymi.
 * Umożliwia wyświetlanie zawartości tablic w konsoli oraz
 * konwertowanie tablic na tablice z innym typem danych liczbowych.
 * @author Wiesław Rychlicki
 * @version 1.0 (2012-02-06)
 */
public class MyRandomArray {

/** Zwraca <code>n</code>-elementową tablicę liczb całkowitych
 * pseudolosowych typu <code>int</code> z zakresu od <code>0</code>
 * do <code>m-1</code>.
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param m liczba naturalna, określająca zakres losowanych wartości
 * (od <code>0</code> do <code>m-1</code>).
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 * @throws java.lang.IllegalArgumentException, gdy podany zakres
 * (<code>m</code>) nie jest liczbą dodatnią.
 */
    public static int[] rndArray(int n, int m) {
        Random rnd = new Random();
        int[] tmp = new int[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = rnd.nextInt(m);
        return tmp;
    }

/** Zwraca posortowaną <code>n</code>-elementową tablicę liczb
 * całkowitych pseudolosowych typu <code>int</code> z zakresu od
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
     * <code>0</code> do <code>m-1</code>.
     * @param n liczba naturalna, rozmiar zwracanej tablicy,
     * @param m liczba naturalna, określająca zakres losowanych wartości
     * (od <code>0</code> do <code>m-1</code>).
     * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
     * tablicy (<code>n</code>) jest liczbą ujemną.
     * @throws java.lang.IllegalArgumentException, gdy podany zakres
     * (<code>m</code>) nie jest liczbą dodatnią.
     */
        public static int[] rndSortArray(int n, int m) {
            Random rnd = new Random();
            int[] tmp = new int[n];
            for(int i = 0; i < n; ++i)
                tmp[i] = rnd.nextInt(m);
            Arrays.sort(tmp);
            return tmp;
        }

    /** Zwraca <code>n</code>-elementową tablicę liczb całkowitych
     * pseudolosowych typu <code>int</code> o niepowtarzających się
     * wartościach z zakresu od <code>0</code> do <code>m-1</code>.
     * @param n liczba naturalna, rozmiar zwracanej tablicy,
     * @param m liczba naturalna określająca zakres losowanych wartości
     * (od <code>0</code> do <code>m-1</code>).
     * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
     * tablicy (<code>n</code>) jest liczbą ujemną.
     * @throws java.lang.IllegalArgumentException, gdy podany zakres
     * (<code>m</code>) nie jest liczbą dodatnią.
     * @throws java.lang.IllegalArgumentException, gdy liczba elementów
     * tablicy jest większa od liczby możliwych wartości
     * (<code>n &rt; m</code>).
     */
        public static int[] rndUniqueArray(int n, int m) {
            if (m < n)
                throw new IllegalArgumentException("Liczba elementów większa
                    od liczby dopuszczalnych wartości!");
            int[] tmp = new int[n];
            Random rnd = new Random();
            int j = 0;
            tmp[j++] = rnd.nextInt(m);
            do {
                int k = rnd.nextInt(m);
                boolean jest = false;
                for(int i = 0; i < j; ++i)
                    if (k == tmp[i]) jest = true;
                if (!jest)
                    tmp[j++] = k;
            } while (j < n);
            return tmp;
        }

    /** Zwraca posortowaną, <code>n</code>-elementową tablicę liczb
     * całkowitych pseudolosowych typu <code>int</code> o niepowtarzających
     * się wartościach z zakresu od <code>0</code> do
     * <code>m-1</code>.
     * @param n liczba naturalna, rozmiar zwracanej tablicy,
     * @param m liczba naturalna, określająca zakres losowanych wartości
     * (od <code>0</code> do <code>m-1</code>).
     * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
     * tablicy (<code>n</code>) jest liczbą ujemną.
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
     * @throws java.lang.IllegalArgumentException, gdy podany zakres
     * (<code>m</code>) nie jest liczbą dodatnią.
     * @throws java.lang.IllegalArgumentException, gdy liczba elementów
     * tablicy jest większa od liczby możliwych wartości
     * (<code>n &rt; m</code>).
     */
        public static int[] rndSortUniqueArray(int n, int m) {
            if (m < n)
                throw new IllegalArgumentException("Liczba elementów większa
                    od liczby dopuszczalnych wartości!");
            int[] tmp = new int[n];
            Random rnd = new Random();
            int j = 0;
            tmp[j++] = rnd.nextInt(m);
            do {
                int k = rnd.nextInt(m);
                boolean jest = false;
                for(int i = 0; i < j; ++i)
                    if (k == tmp[i]) jest = true;
                if (!jest)
                    tmp[j++] = k;
            } while (j < n);
            Arrays.sort(tmp);
            return tmp;
        }

    /** Wyświetla liczby zawarte w tablicy w konsoli, oddzielając kolejne
     * wartości odstępem.
     * @param tab jednowymiarowa tablica liczb całkowitych.
     */
        public static void arrayPrint(int[] tab) {
            int n = tab.length;
            for(int i = 0; i < n; ++i)
                System.out.print(tab[i]+" ");
        }

    /** Wyświetla liczby zawarte w tablicy w konsoli, oddzielając kolejne
     * wartości odstępem. Po wyświetleniu ostatniej liczby przesyłany jest
     * do konsoli znak końca linii.
     * @param tab jednowymiarowa tablica liczb całkowitych.
     */
        public static void arrayPrintln(int[] tab) {
            int n = tab.length;
            for(int i = 0; i < n; ++i)
                System.out.print(tab[i]+" ");
            System.out.println();
        }

    /** Dodaje do wszystkich liczb zawartych w tablicy podaną liczbę
     * całkowitą.
     * @param tab jednowymiarowa tablica liczb całkowitych typu
     * <code>int</code>.
     * @param d liczba całkowita typu <code>int</code>.
     */
        public static void addToArray(int[] tab, int d) {
            int n = tab.length;
            for(int i = 0; i < n; ++i)
                tab[i] += d;
        }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zwraca <code>n</code>-elementową tablicę liczb zmiennoprzecinkowych
 * pseudolosowych typu <code>double</code> z określonego przedziału.
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 */
    public static double[] rndArray(int n, double a) {
        Random rnd = new Random();
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = a*rnd.nextDouble();
        return tmp;
    }

/** Zwraca <code>n</code>-elementową tablicę liczb zmiennoprzecinkowych
 * pseudolosowych typu <code>double</code> z określonego przedziału,
 * zaokrąglonych z podaną precyzją.
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @param prec liczba miejsc po przecinku.
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 */
    public static double[] rndArray(int n, double a, int prec) {
        Random rnd = new Random();
        double wsp = Math.pow(10.0, prec);
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (int)(a*rnd.nextDouble()*wsp+0.5)/wsp;
        return tmp;
    }

/** Zwraca <code>n</code>-elementową tablicę liczb zmiennoprzecinkowych
 * pseudolosowych typu <code>double</code> z określonego przedziału,
 * różniących się o wielokrotność podanego kroku (wartość dodatnia).
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @param step krok - najmniejsza możliwa różnica pomiędzy liczbami
 * w tablicy (różnice pomiędzy wartościami liczb w tablicy są
 * wielokrotnością parametru <code>step</code>).
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 * @throws java.lang.IllegalArgumentException, gdy podany parametr
 * <code>step</code> nie jest liczbą dodatnią.
 */
    public static double[] rndArray(int n, double a, double step) {
        Random rnd = new Random();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        int k = (int)(a/step)+1;
        double wsp = Math.pow(10.0, 1+(int)Math.log10(1/step));
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = (int)(step*rnd.nextInt(k)*wsp+0.5)/wsp;
        return tmp;
    }

/** Zwraca <code>n</code>-elementową tablicę liczb zmiennoprzecinkowych
 * pseudolosowych typu <code>double</code> o niepowtarzających się wartościach
 * z określonego przedziału,
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 * @throws java.lang.IllegalArgumentException, gdy podany parametr
 * <code>a</code> jest zerem.
 */
    public static double[] rndUniqueArray(int n, double a) {
        if (a == 0)
            throw new IllegalArgumentException("a = 0.0");
        double[] tmp = new double[n];
        Random rnd = new Random();
        int j = 0;
        tmp[j++] = a*rnd.nextDouble();
        do {
            double x = a*rnd.nextDouble();
            boolean jest = false;
            for(int i = 0; i < j; ++i)
                if (x == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = x;
        } while (j < n);
        return tmp;
    }

/** Zwraca posortowaną, <code>n</code>-elementową tablicę liczb
 * zmiennoprzecinkowych pseudolosowych typu <code>double</code>
 * z określonego przedziału.
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 */
    public static double[] rndSortArray(int n, double a) {
        Random rnd = new Random();
        double[] tmp = new double[n];
        for(int i = 0; i < n; ++i)
            tmp[i] = a*rnd.nextDouble();
        Arrays.sort(tmp);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        return tmp;
    }

/** Zwraca posortowaną, <code>n</code>-elementową tablicę liczb
 * zmiennoprzecinkowych pseudolosowych typu <code>double</code>
 * z określonego przedziału, o niepowtarzających się wartościach.
 * @param n liczba naturalna, rozmiar zwracanej tablicy,
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>
 * określająca przedział losowanych wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy tablicę wypełnioną zerami.
 * @throws java.lang.NegativeArraySizeException, gdy podany wymiar
 * tablicy (<code>n</code>) jest liczbą ujemną.
 * @throws java.lang.IllegalArgumentException, gdy podany parametr
 * <code>a</code> jest zerem.
 */
    public static double[] rndSortUniqueArray(int n, double a) {
        if (a == 0)
            throw new IllegalArgumentException("a = 0.0");
        double[] tmp = new double[n];
        Random rnd = new Random();
        int j = 0;
        tmp[j++] = a*rnd.nextDouble();
        do {
            double x = a*rnd.nextDouble();
            boolean jest = false;
            for(int i = 0; i < j; ++i)
                if (x == tmp[i]) jest = true;
            if (!jest)
                tmp[j++] = x;
        } while (j < n);
        Arrays.sort(tmp);
        return tmp;
    }

/** Wyświetla liczby zawarte w tablicy w konsoli, oddzielając kolejne
 * wartości odstępem.
 * @param tab jednowymiarowa tablica liczb typu <code>double</code>.
 */
    public static void arrayPrint(double[] tab) {
        int n = tab.length;
        for(int i = 0; i < n; ++i)
            System.out.print(tab[i]+" ");
    }

/** Wyświetla liczby zawarte w tablicy w konsoli, oddzielając kolejne
 * wartości odstępem. Po wyświetleniu ostatniej liczby przesyłany jest
 * do konsoli znak końca linii.
 * @param tab jednowymiarowa tablica liczb typu <code>double</code>.
 */
    public static void arrayPrintln(double[] tab) {
        int n = tab.length;
        for(int i = 0; i < n; ++i)
            System.out.print(tab[i]+" ");
        System.out.println();
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Wyświetla sformatowane liczby (zapisane w tablicy) w konsoli. Sposób
 * formatowania określony jest z wykorzystaniem parametru <code>spec</code>
 * zbudowanego według zasad zdefiniowanych w klasie
 * <a href="http://docs.oracle.com/javase/1.5.0/docs/api/java/util/
    Formatter.html">.
 * Formatter</a>, np.: <code>"%.2f "</code> - wyświetla liczbę
 * zmiennoprzecinkową z dwoma miejscami po przecinku.
 * @param spec łańcuch znaków określający format liczby.
 * @param tab jednowymiarowa tablica liczb typu <code>double</code>.
 */
   public static void arrayPrintf(String spec, double[] tab) {
       int n = tab.length;
       for(int i = 0; i < n; ++i)
           System.out.printf(spec, tab[i]);
   }

/** Dodaje do wszystkich liczb zawartych w tablicy podaną liczbę
 * zmiennoprzecinkową typu <code>double</code>.
 * @param tab jednowymiarowa tablica liczb zmiennoprzecinkowych typu
 * <code>double</code>,
 * @param d liczba zmiennoprzecinkowa typu <code>double</code>.
 */
   public static void addToArray(double[] tab, double d) {
       int n = tab.length;
       for(int i = 0; i < n; ++i)
           tab[i] += d;
   }

/** Zaokrągla wszystkie liczby zawarte w tablicy z określoną precyzją.
 * @param tab jednowymiarowa tablica liczb zmiennoprzecinkowych typu
 * <code>double</code>,
 * @param prec liczba całkowita typu <code>int</code>, precyzja
 * zaokrąglenia.
 * @see #round
 */
   public static void roundArray(double[] tab, int prec) {
       int n = tab.length;
       double wsp = Math.pow(10.0, prec);
       for(int i = 0; i < n; ++i)
           tab[i] = (int)(tab[i]*wsp+0.5)/wsp;
   }

/** Zwraca liczbę zmiennoprzecinkową typu <code>double</code> zaokrągloną
 * z określoną precyzją.
 * @param x liczba zmiennoprzecinkowa typu <code>double</code>.
 * @param prec liczba całkowita typu <code>int</code> określająca
 * precyzję zaokrąglenia:
 * <br><code>prec &gt; 0</code> - liczba miejsc po przecinku,
 * <br><code>prec = 0</code> - zaokrąglenie do wartości całkowitej,
 * <br><code>prec &lt; 0</code> - wartość bezwzględna liczby
 * <code>prec</code> określa liczbę zer przed przecinkiem, zaokrąglenie
 * do pełnych dziesiątek (<code>-1</code>), setek (<code>-2</code>) itd.
 */
   public static double round(double x, int prec) {
       double wsp = Math.pow(10.0, prec);
       return(int)(x*wsp+0.5)/wsp;
   }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zwraca liczbę zmiennoprzecinkową typu <code>float</code> zaokrągloną
 * z określoną precyzją.
 * @param x liczba zmiennoprzecinkowa typu <code>float</code>.
 * @param prec liczba całkowita typu <code>int</code> określająca
 * precyzję zaokrąglenia:
 * <br><code>prec &gt; 0</code> - liczba miejsc po przecinku,
 * <br><code>prec = 0</code> - zaokrąglenie do wartości całkowitej,
 * <br><code>prec &lt; 0</code> - wartość bezwzględna liczby
 * <code>prec</code> określa liczbę zer przed przecinkiem, zaokrąglenie
 * do pełnych dziesiątek (<code>-1</code>), setek (<code>-2</code>) itd.
 */
    public static float round(float x, int prec) {
        float wsp = (float) Math.pow(10.0, prec);
        return (int)(x*wsp+0.5)/wsp;
    }

/** Zwraca pseudolosową liczbę zmiennoprzecinkową typu <code>double</code>
 * z określonego przedziału, zaokrągloną z podaną precyzją.
 * @param a liczba zmiennoprzecinkowa typu <code>double</code>,
 * określająca przedział losowanej wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy wartość <code>0.0</code>.
 * @param prec precyzja zaokrąglenia.
 * @see #round
 */
    public static double random(double a, int prec) {
        return round(a*Math.random(), prec);
    }

/** Zwraca pseudolosową liczbę zmiennoprzecinkową typu <code>float</code>
 * z określonego przedziału, zaokrągloną z podaną precyzją.
 * @param a liczba zmiennoprzecinkowa typu <code>float</code>,
 * określająca przedział losowanej wartości:
 * <br> <code>&lt;0, a)</code> - dla <code>a &gt 0</code>,
 * <br> <code>(a, 0&gt</code> - dla <code>a &lt 0</code>.
 * <br> Dla <code>a = 0</code> otrzymamy wartość <code>0.0f</code>.
 * @param prec precyzja zaokrąglenia.
 * @see #round
 */
    public static float random(float a, int prec) {
        return (float) round(a*Math.random(), prec);
    }
}
```

20. Działania na ułamkach — budujemy klasę Fraction

Zadanie 20.1. Z20_1.java, Fraction.java

Zacznijmy od zdefiniowania klasy `Fraction` z dwoma prywatnymi polami `num` (ang. *numerator* — licznik) i `den` (ang. *denominator* — mianownik) typu całkowitego `int`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Klasa <code>Fraction</code> służy do wykonywania podstawowych działań
 * na ułamkach zwykłych (ang. <i>fraction</i> - ułamek).
 * @author Wiesław Rychlicki
 * @version 1.0 (2012-02-07)
 */
public class Fraction {
/** Licznik ułamka (<i>numerator</i> - licznik). */
    private int num;
/** Mianownik ułamka (<i>denominator</i> - mianownik). */
    private int den;
}
```

Do kodu klasy dodamy *konstruktor*, czyli metodę o takiej samej nazwie (`Fraction`) jak nazwa klasy. Konstruktor nie zwraca żadnej wartości — jego zadaniem jest inicjowanie pól obiektu. Konstruktor jest wywoływany automatycznie podczas tworzenia obiektu za pomocą operatora `new`. Znaczenie parametrów `m` i `n` jest objaśnione w komentarzu dokumentacyjnym.

```
/** Tworzy nowy obiekt <code>Fraction</code> - ułamek w postaci
 * <code>m/n</code> - na podstawie wartości całkowitych parametrów
 * <code>m</code> i <code>n</code>.
 * @param m liczba całkowita (licznik ułamka),
 * @param n liczba całkowita (mianownik ułamka).
 * @throws IllegalArgumentException, gdy podamy mianownik n = 0.
 */
    Fraction (int m, int n) {
        this.num = m;
        if (n != 0)
            this.den = n;
        else
            throw new IllegalArgumentException("Parametr n = 0!");
    }
```

Podstawienie wartości parametrów do pól obiektu nastąpiło przy użyciu słowa kluczowego `this`, które umożliwia dostęp do zmiennych egzemplarzowych i metod bieżącej klasy (`this.num = m; this.den = n;`). Ponieważ w tym przykładzie nazwy identyfikatorów (parametrów i pól klasy) nie powtarzają się, wykorzystanie słowa `this` nie było konieczne — wystarczyłoby podstawienie w postaci: `num = m; den = n;`. Podanie mianownika równego 0 spowoduje zwrócenie wyjątku `IllegalArgumentException`.

Metoda `toString()` jest zaimplementowana w klasie `Object` i wskazane jest jej przysłonięcie w każdej nowo tworzonej klasie. Użycie operatora + do łączenia (*konkatenacji*) łańcucha z danymi innego typu powoduje zamianę tych danych na łańcuchy znaków i połączenie łańcuchów w jeden łańcuch. W przypadku dodawania obiektów wykorzystywana jest w tym celu metoda `toString()` przesłaniająca metodę `Object.toString()`.

```
/** Zwraca łańcuch znaków reprezentujący ten obiekt
 * <code>Fraction</code>. Przykład zwracanego łańcucha
 * <code>"3/4"</code>, <code>"-5/25"</code> itp.
 */
    @Override public String toString() {
        return num+"/"+den;
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Utworzoną klasę pokażemy w prostej aplikacji.

```
/** <h3>Zadanie Z20.1</h3>
 * Tworzenie konstruktora <code>Fraction(int, int)</code>
 * i metody <code>toString()</code>.
 */
public class Z20_1 {
/** Prezentuje działanie konstruktora <code>Fraction(int, int)</code>
 * i metody <code>toString()</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(3, 4);
        System.out.println("a = "+a);
    }
}
```

Polecenie `Fraction a = new Fraction(3, 4)` tworzy nowy obiekt `a` klasy `Fraction` i nadaje mu wartość ułamka *¾*.

Operacja dodawania `"a = "+a` (łańcucha znaków `"a = "` i obiektu `a`) jest równoważna operacji dodawania `"a = "+a.toString()` i pokazuje działanie metody `toString()`.

Zwróćmy uwagę na użyte komentarze dokumentacyjne. W ten sposób będziemy tworzyli dokumentację klasy `Fraction` i wszystkich programów demonstrujących działanie budowanej klasy. Polecenie tworzące dokumentację wszystkich klas zapisanych w bieżącym folderze może mieć postać: `javadoc -d Fraction -author -version -nodeprecatedlist -nohelp -private -linksource *.java`.


Uwaga

> Opcja `-linksource` powoduje utworzenie w dokumentacji linków do stron zawierających kod źródłowy klas (zapisany w postaci stron HTML).

Zadanie 20.2. Z20_2.java, Fraction.java

Bezparametrowy konstruktor tworzy obiekt odpowiadający liczbie *0* (`num = 0; den = 1;`).

```
/** Tworzy nowy obiekt <code>Fraction</code> - ułamek w postaci
 * <code>0/1</code> o wartości odpowiadającej liczbie całkowitej
 * <code>0</code>.
 */
    Fraction() {
        this.num = 0;
        this.den = 1;
    }
```

Ten konstruktor również mógłby mieć postać `Fraction() {this(0, 1);}` — używając słowa kluczowego `this`, wywołamy w ten sposób konstruktor z dwoma parametrami i zbudujemy obiekt reprezentujący ułamek $^{0}/_{1}$.

Konstruktor z jednym parametrem tworzy ułamek o wartości odpowiadającej liczbie całkowitej (mianownik tego ułamka jest równy *1*, a licznik jest równy podanej liczbie całkowitej).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Tworzy nowy obiekt <code>Fraction</code> - ułamek w postaci
 * <code>m/1</code> o wartości odpowiadającej liczbie całkowitej
 * <code>m</code>.
 * @param m liczba całkowita (licznik ułamka).
 */
    Fraction(int m) {
        this.num = m;
        this.den = 1;
    }
```

Również w tym przypadku możemy skonstruować obiekt, używając słowa kluczowego `this` konstruktora z dwoma parametrami: `Fraction(int m) {this(m, 1);}`.

Działanie konstruktorów pokażemy w aplikacji:

```
/** <h3>Zadanie Z20.2</h3>
 * Tworzenie konstruktorów <code>Fraction()</code>
 * i <code>Fraction(int)</code>.
 */
public class Z20_2 {
/** Prezentuje działanie bezparametrowego konstruktora
 * <code>Fraction()</code> i konstruktora z jednym parametrem
 * <code>Fraction(int)</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction();
        System.out.println("a = "+a);
        System.out.println(new Fraction(4));
    }
}
```

Zadanie 20.3. Z20_3.java, Fraction.java

Operator podstawiania (w języku Java, w stosunku do obiektów) = przypisuje jednemu obiektowi referencję do drugiego obiektu. Jeśli chcemy utworzyć inny obiekt o takiej samej wartości, to musimy posłużyć się konstruktorem kopiującym:

```
/** Konstruktor kopiujący - tworzy nowy obiekt <code>Fraction</code>
 * o wartości równej wartości obiektu <code>x</code> klasy
 * <code>Fraction</code> podanego jako parametr.
 * @param x obiekt klasy <code>Fraction</code>.
 */
    Fraction (Fraction x) {
        this.num = x.num;
        this.den = x.den;
    }
```

Tworzymy obiekt `a` (`Fraction a = new Fraction(3, 11)`) i drugi obiekt, `b`, będący jego kopią (`Fraction b = new Fraction(a)`). Obiekty reprezentują ten sam ułamek $^{3}/_{11}$. Operator == porównuje referencje obiektów, a nie ich wartości. Wyrażenie `a == b` przyjmuje wartość `false`.

Deklarujemy zmienną `c` typu obiektowego i przypisujemy jej referencję do obiektu `a` (`Fraction c = a`) — zmienne `a` i `c` reprezentują ten sam obiekt (i tę samą wartość $^{3}/_{11}$). Wyrażenie `a == c` przyjmuje wartość `true`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** <h3>Zadanie Z20.3</h3>
 * Tworzenie konstruktora kopiującego <code>Fraction(Fraction)</code>.
 */
public class Z20_3 {
/** Prezentuje działanie konstruktora kopiującego
 * <code>Fraction(Fraction)</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(3, 11);
        Fraction b = new Fraction(a);
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("Porównanie a == b, wynik: "+(a == b));
        Fraction c = a;
        System.out.println("c = "+c);
        System.out.println("Porównanie a == c, wynik: "+(a == c));
    }
}
```

Zadanie 20.4. Z20_4.java, Fraction.java

W przypadku gdy mianownik ułamka jest liczbą ujemną, należy zmienić znak mianownika i licznika ułamka. W kodzie metody pominięto słowo kluczowe `this`. Nie powoduje to żadnych problemów, a kod będzie czytelniejszy.

```
/** Koryguje wartość pól obiektu <code>Fraction</code> w taki sposób, aby
 * mianownik był zawsze liczbą dodatnią.
 */
    private void correction() {
        if (den < 0) {
           den = -den;
           num = -num;
       }
    }
```

Wywołanie tej metody umieścimy w kodzie konstruktora:

```
Fraction (int m, int n) {
    this.num = m;
    if (n != 0)
        this.den = n;
    else
        throw new IllegalArgumentException("Parametr n = 0!");
    this.correction();
}
```

Działanie metody przetestujemy, budując różne ułamki:

```
/** <h3>Zadanie Z20.4</h3>
 * Tworzenie prywatnej metody <code>correction()</code>, poprawiającej
 * znaki licznika i mianownika ułamka w taki sposób, aby mianownik był
 * zawsze dodatni.
 */
public class Z20_4 {
/** Prezentuje działanie prywatnej metody <code>correction()</code>
 * w kodzie konstruktora.
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        System.out.println("-3/4 = "+new Fraction(-3, 4));
        System.out.println("3/-4 = "+new Fraction(3, -4));
        System.out.println("2/5 = "+new Fraction(2, 5));
        System.out.println("-2/-5 = "+new Fraction(-2, -5));
    }
}
```

Zadanie 20.5. Z20_5.java, Fraction.java

Metodę zwracającą największy wspólny dzielnik możemy zbudować na podstawie rozwiązania zadania 15.8. Przyjmujemy zgodnie z definicją *nwd(0, n) = n* (przypadek *nwd(m, 0)* w naszej klasie nie wystąpi — mianownik jest zawsze dodatni). Ponadto dla $m < 0$ zmienimy znak liczby (`if (m < 0) m = -m;`) i wykonamy obliczenia dla $|m|$.

```
/** Zwraca największy wspólny dzielnik pary liczb całkowitych dodatnich
 * <code>m</code> i <code>n</code>, podanych jako parametry wywołania
 * metody.
 * </br><b>Uwaga:</b> <i>Podanie ujemnych wartości tych parametrów nie
 * spowoduje błędu; znaki liczb zostaną zignorowane - liczby zostaną
 * zastąpione ich bezwzględnymi wartościami</i>.
 * @param m liczba całkowita dodatnia,
 * @param n liczba całkowita dodatnia.
 */
    private int nwd(int m, int n) {
        if (m == 0)
            return n;
        if (m < 0) m = -m;
        while (m != n)
            if (m > n)
                m -= n;
            else
                n -= m;
        return m;
    }
```

Metoda `reduce()` oblicza największy wspólny dzielnik dla licznika i mianownika ułamka reprezentowanego przez obiekt (`int d = nwd(num, den)`). Jeśli dzielnik `d` jest różny od jedności, to dzielimy licznik i mianownik przez ten dzielnik.

```
/** Skraca ułamek reprezentowany przez obiekt przez największy wspólny
 * dzielnik licznika i mianownika.
 */
    public Fraction reduce() {
        int d = nwd(num, den);
        if (d != 1) {
            this.den /= d;
            this.num /= d;
        }
        return this;
    }
```

Tę metodę będziemy później stosować w metodach wykonujących działania na ułamkach.

Ułamek możemy skracać przez dodatni dzielnik `d` podany jako parametr.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Skraca ułamek reprezentowany przez obiekt przez podany dzielnik
 * <code>d</code>, o ile jest to możliwe.
 * @param d liczba całkowita dodatnia typu <code>int</code>.
 * @throws IllegalArgumentException, gdy podamy dzielnik
 * <code>d &le; 0</code>.
 */
    public Fraction reduce(int d) {
        if (d <= 0)
            throw new IllegalArgumentException("Parametr d <= 0!");
        if (this.num%d == 0 && this.den%d ==0) {
            this.den /= d;
            this.num /= d;
        }
        return this;
    }
```

Rozszerzanie ułamka nie powoduje zmiany jego wartości i polega na pomnożeniu licznika i mianownika ułamka przez tę samą liczbę różną od zera.

```
/** Rozszerza ułamek reprezentowany przez obiekt przez podaną liczbę
 * całkowitą <code>n &ne; 0</code>.
 * @param n liczba całkowita typu <code>int</code> różna od 0.
 * @throws IllegalArgumentException, gdy podamy <code>n = 0</code>.
 */
    public Fraction equivalent(int n) {
        if (n == 0)
            throw new IllegalArgumentException("Parametr n = 0!");
        this.den *= n;
        this.num *= n;
        this.correction(); // mianownik powinien pozostać dodatni
        return this;
    }
```

Działanie metody przetestujemy, budując różne ułamki:

```
/** <h3>Zadanie Z20.5</h3>
 * Tworzenie prywatnej metody <code>nwd(int, int)</code> obliczającej
 * największy wspólny dzielnik pary liczb dodatnich oraz publicznej
 * metody (<code>reduce()</code> skracającej ułamek reprezentowany przez
 * obiekt.
 */
public class Z20_5 {
/** Prezentuje działanie metody <code>reduce()</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(3, 4);
        System.out.println("a = "+a+" = "+a.reduce());
        a = new Fraction(15, 20);
        System.out.println("a = "+a+" = "+a.reduce());
        a = new Fraction(-9, -18);
        System.out.println("a = "+a+" = "+a.reduce());
        a = new Fraction(0, 5);
        System.out.println("a = "+a+" = "+a.reduce());
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.6. Z20_6.java, Fraction.java

Mnożenie ułamków możemy przedstawić prostym wzorem $\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$. Obliczamy iloczyn liczników ułamków i iloczyn mianowników, a następnie budujemy z tych iloczynów nowy ułamek (obiekt `Fraction`).

```
/** Zwraca nowy obiekt <code>Fraction</code> - iloczyn ułamka
 * reprezentowanego przez obiekt wywołujący metodę i ułamka
 * reprezentowanego przez obiekt przekazany jako parametr.
 * @param x ułamek, obiekt klasy <code>Fraction</code>.
 */
    public Fraction mult(Fraction x) {
        return new Fraction(this.num*x.num, this.den*x.den).reduce();
    }
```

Mnożenie ułamka przez liczbę całkowitą wykonujemy podobnie $\frac{a}{b} \cdot m = \frac{a \cdot m}{b}$.

```
/** Zwraca nowy obiekt <code>Fraction</code> - iloczyn ułamka
 * reprezentowanego przez obiekt wywołujący metodę i liczby całkowitej
 * typu <code>int</code> przekazanej jako parametr.
 * @param m liczba całkowita typu <code>int</code>.
 */
    public Fraction mult(int m) {
        return new Fraction(this.num*m, this.den).reduce();
    }
```

Możemy zdefiniować mnożenie ułamka przez liczbę naturalną przy użyciu konstruktora jednoargumentowego i wyżej określonego mnożenia ułamków:

```
public Fraction mult(int m) {
    return this.mult(new Fraction(m));
}
```

lub:

```
public Fraction mult(int m) {
    return new Fraction(m).mult(this);
}
```

W celu sprawdzenia zbudowanych metod wykonamy kilka przykładowych mnożeń.

```
/** <h3>Zadanie Z20.6</h3>
 * Tworzenie metod <code>mult(Fraction)</code> i <code>mult(int)</code>,
 * zwracających nowy obiekt będący iloczynem ułamka reprezentowanego
 * przez obiekt i drugiego ułamka lub liczby całkowitej.
 */
public class Z20_6 {
/** Prezentuje działanie metod zwracających obiekt reprezentujący iloczyn
 * ułamka przez ułamek i ułamka przez liczbę.
 */
    public static void main(String[] args) {
        System.out.println("3/4 * 8/9 = "+new Fraction(3, 4).
            mult(new Fraction(8, 9)));
        Fraction a = new Fraction(-5, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("b = "+b);
        System.out.println("a*b = "+a.mult(b));
        System.out.println("b*a = "+b.mult(a));
        System.out.println("a*2 = "+a.mult(2));
        System.out.println("a*(-14) = "+a.mult(-14));
    }
}
```

Zadanie 20.7. Z20_7.java, Fraction.java

Mnożenie dwóch ułamków (obiektów `Fraction`):

```
/** Zwraca obiekt klasy <code>Fraction</code> będący iloczynem dwóch
 * obiektów <code>Fraction</code>.
 * @param x obiekt klasy <code>Fraction</code>,
 * @param y obiekt klasy <code>Fraction</code>.
 */
    public static Fraction prod(Fraction x, Fraction y) {
        return x.mult(y);
    }
```

Mnożenie ułamka (obiektu `Fraction`) przez liczbę całkowitą lub mnożenie liczby przez ułamek:

```
/** Zwraca obiekt klasy <code>Fraction</code> będący iloczynem obiektu
 * <code>Fraction</code> i liczby całkowitej typu <code>int</code>.
 * @param x obiekt klasy <code>Fraction</code>,
 * @param m liczba całkowita typu <code>int</code>.
 */
    public static Fraction prod(Fraction x, int m) {
        return x.mult(m);
    }

/** Zwraca obiekt klasy <code>Fraction</code> będący iloczynem liczby
 * całkowitej typu <code>int</code> i obiektu <code>Fraction</code>.
 * @param m liczba całkowita typu <code>int</code>.
 * @param x obiekt klasy <code>Fraction</code>.
 */
    public static Fraction prod(int m, Fraction x) {
        return x.mult(m);
    }
```

Iloczyn dwóch liczb całkowitych zapisany w postaci obiektu klasy `Fraction`:

```
/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący iloczyn
 * dwóch liczb całkowitych typu <code>int</code>.
 * @param m liczba całkowita typu <code>int</code>,
 * @param n liczba całkowita typu <code>int</code>.
 */
    public static Fraction prod(int m, int n) {
        return new Fraction(m).mult(n);
    }
```

Kilka przykładowych iloczynów ilustruje działanie utworzonych metod statycznych.

```
/** <h3>Zadanie Z20.7</h3>
 * Tworzenie metod statycznych realizujących mnożenie dwóch obiektów
 * <code>Fraction</code>, obiektu <code>Fraction</code>
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
 * i liczby całkowitej oraz mnożenie dwóch liczb całkowitych z wynikiem
 * typu <code>Fraction</code>.
 */
public class Z20_7 {
/** Prezentuje działanie metod statycznych obliczających iloczyn
 * ułamka przez ułamek, ułamka przez liczbę całkowitą i iloczyn dwóch
 * liczb całkowitych z wynikami typu <code>Fraction</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a*b = "+Fraction.prod(a, b));
        System.out.println("a*4 = "+Fraction.prod(a, 4));
        System.out.println("-3*b = "+Fraction.prod(-3, b));
        System.out.println("2*3 = "+Fraction.prod(2, 3));
    }
}
```

Zadanie 20.8. Z20_8.java, Fraction.java

Utworzenie ułamka odwrotnego jest prostą czynnością — wystarczy zamienić wartość licznika i mianownika. Ułamek w postaci $^{0}/_{1}$ (liczba *0*) nie ma odwrotności, więc zamiana licznika i mianownika spowoduje próbę utworzenia ułamka $^{1}/_{0}$ (`new Fraction(1, 0)`) — konstruktor zgłosi wyjątek `IllegalArgumentException`.

```
/** Zwraca obiekt <code>Fraction</code> reprezentujący odwrotność ułamka
 * zawartego w obiekcie wywołującym tę metodę.
 * @throws IllegalArgumentException, gdy ten obiekt reprezentuje ułamek
 * <code>0/1</code>.
 */
    public Fraction multInv() {
        return new Fraction(this.den, this.num);
    }
```

Podobnie działa metoda statyczna:

```
/** Zwraca obiekt <code>Fraction</code> reprezentujący odwrotność ułamka
 * zawartego w obiekcie <code>x</code> podanym jako parametr.
 * @param x obiekt klasy <code>Fraction</code>.
 * @throws IllegalArgumentException, gdy obiekt <code>x</code>
 * reprezentuje ułamek <code>0/1</code>.
 */
    public static Fraction multInv(Fraction x) {
        return new Fraction(x.den, x.num);
    }
```

Metody przetestujemy, obliczając odwrotności kilku ułamków. Zwróćmy uwagę na możliwość wykonywania dzielenia ułamków według szkolnej regułki: *aby podzielić przez ułamek, mnożymy przez jego odwrotność*.

```
/** <h3>Zadanie Z20.8</h3>
 * Tworzenie metod wyznaczających odwrotność ułamka.
 */
public class Z20_8 {
/** Prezentuje działanie metod obliczających odwrotność ułamka.
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        System.out.println("1/a = "+a.multInv());
        Fraction b = new Fraction(-7, 9);
        System.out.println("b = "+b);
        System.out.println("1/b = "+Fraction.multInv(b));
        Fraction c = new Fraction();
        System.out.println("c = "+c);
        try {
            System.out.println("1/c = "+Fraction.multInv(c));
        } catch(IllegalArgumentException e) {
            System.out.println("Nie istnieje odwrotność 0!");
        }
        System.out.println("a/b = "+a.mult(b.multInv()));
        System.out.println("a/b = "+b.multInv().mult(a));
        System.out.println("a/b = "+Fraction.
            prod(a,Fraction.multInv(b)));
     }
}
```

Zadanie 20.9. Z20_9.java, Fraction.java

Do klasy `Fraction` dołączymy metody wykonujące dzielenie ułamka przez ułamek i dzielenie ułamka przez liczbę całkowitą.

```
/** Zwraca nowy obiekt <code>Fraction</code> - iloraz ułamka zawartego
 * w obiekcie wywołującym metodę (dzielna) i ułamka reprezentowanego
 * przez obiekt przekazany jako parametr (dzielnik).
 * @param x ułamek - obiekt klasy <code>Fraction</code> (dzielnik).
 * @throws ArithmeticException, gdy obiekt <code>x</code> reprezentuje
 * <code>0/1</code>.
 */
    public Fraction div(Fraction x) {
        if (x.num == 0)
            throw new ArithmeticException("Dzielenie przez 0!");
        return new Fraction(this.num*x.den, this.den*x.num).reduce();
    }

/** Zwraca nowy obiekt <code>Fraction</code> - iloraz ułamka zawartego
 * w obiekcie wywołującym metodę (dzielna) i liczby całkowitej typu
 * <code>int</code> przekazanej jako parametr (dzielnik).
 * @param m liczba całkowita typu <code>int</code> (dzielnik).
 * @throws ArithmeticException, gdy podamy <code>m = 0</code>.
 */
    public Fraction div(int m) {
        if (m == 0)
            throw new ArithmeticException("Dzielenie przez 0!");
        return new Fraction(this.num, this.den*m).reduce();
    }
```

Działanie metod sprawdzimy, wykonując kilka przykładowych dzieleń. Zwróćmy uwagę na przypadki dzielenia przez *0* — następuje zwrócenie wyjątku przez metodę i jego obsługa w programie.

```
/** <h3>Zadanie Z20.9</h3>
 * Tworzenie metod <code>div(Fraction)</code> i <code>div(int)</code>
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
 * zwracających nowy obiekt będący ilorazem ułamka reprezentowanego
 * przez obiekt i drugiego ułamka lub liczby całkowitej.
 */
public class Z20_9 {
/** Prezentuje działanie metod zwracających obiekt reprezentujący iloraz
 * ułamka przez ułamek i ułamka przez liczbę.
 */
    public static void main(String[] args) {
        System.out.println("(3/4) / (3/7) = "+new Fraction(3, 4).
            div(new Fraction(3, 7)));
        Fraction a = new Fraction(-5, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a/b = "+a.div(b));
        System.out.println("b/a = "+b.div(a));
        System.out.println("a/2 = "+a.div(2));
        System.out.println("a/(-4) = "+a.div(-4));
        try {
            Fraction c = new Fraction(0, 5);
            System.out.println("c = "+c);
            System.out.println("a/c = "+a.div(c));
        } catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
        try {
            System.out.println("a/0 = "+a.div(0));
        } catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Zadanie 20.10. Z20_10.java, Fraction.java

Do klasy `Fraction` dołączamy cztery metody statyczne realizujące różne przypadki dzielenia — niezbędne objaśnienia zawarte są w komentarzach dokumentacyjnych.

```
/** Zwraca obiekt klasy <code>Fraction</code> będący ilorazem dwóch
 * obiektów <code>Fraction</code>.
 * @param x obiekt klasy <code>Fraction</code> (dzielna),
 * @param y obiekt klasy <code>Fraction</code> (dzielnik).
 * @throws ArithmeticException, gdy obiekt <code>y</code> reprezentuje
 * <code>0/1</code>.
 */
    public static Fraction quot(Fraction x, Fraction y) {
        if (y.num == 0)
            throw new ArithmeticException("Dzielenie przez 0!");
        return x.div(y);
    }

/** Zwraca obiekt klasy <code>Fraction</code> będący ilorazem obiektu
 * <code>Fraction</code> (dzielna) i liczby całkowitej typu
 * <code>int</code> (dzielnik).
 * @param x obiekt klasy <code>Fraction</code> (dzielna),
 * @param m liczba całkowita typu <code>int</code> (dzielnik).
 * @throws ArithmeticException, gdy podamy <code>m = 0</code>.
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static Fraction quot(Fraction x, int m) {
     if (m == 0)
         throw new ArithmeticException("Dzielenie przez 0!");
        return x.div(m);
    }

/** Zwraca obiekt klasy <code>Fraction</code> będący ilorazem liczby
 * całkowitej typu <code>int</code> (dzielna) i obiektu
 * <code>Fraction</code> (dzielnik).
 * @param m liczba całkowita typu <code>int</code> (dzielna).
 * @param x obiekt klasy <code>Fraction</code> (dzielnik).
 * @throws ArithmeticException, gdy obiekt <code>x</code> reprezentuje
 * <code>0/1</code>.
 */
    public static Fraction quot(int m, Fraction x) {
        if (x.num == 0)
            throw new ArithmeticException("Dzielenie przez 0!");
        return new Fraction(m).div(x);
    }

/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący iloraz
 * dwóch liczb całkowitych typu <code>int</code>.
 * @param m liczba całkowita typu <code>int</code> (dzielna),
 * @param n liczba całkowita typu <code>int</code> (dzielnik).
 * @throws ArithmeticException, gdy podamy <code>n = 0</code>.
 */
    public static Fraction quot(int m, int n) {
        if (n == 0)
            throw new ArithmeticException("Dzielenie przez 0!");
        return new Fraction(m, n).reduce();
    }
```

Kilka przykładowych dzieleń pokazuje działanie przeciążonych metod `quot()` dla różnych par argumentów (dzielenie ułamka przez ułamek, ułamka przez liczbę całkowitą, liczby całkowitej przez ułamek oraz dzielenie dwóch liczb całkowitych z wynikiem typu `Fraction`).

```
/** <h3>Zadanie Z20.10</h3>
 * Tworzenie metod statycznych realizujących dzielenie dwóch obiektów
 * <code>Fraction</code>, obiektu <code>Fraction</code> przez liczbę
 * całkowitą, liczby całkowitej przez obiekt oraz dzielenie dwóch liczb
 * całkowitych z wynikiem typu <code>Fraction</code>.
 */
public class Z20_10 {
/** Prezentuje działanie metod statycznych obliczających iloraz ułamka
 * przez ułamek, ułamka przez liczbę całkowitą, liczby całkowitej przez
 * ułamek i iloraz dwóch liczb całkowitych z wynikami typu
 * <code>Fraction</code>.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a/b = "+Fraction.quot(a, b));
        System.out.println("a/4 = "+Fraction.quot(a, 4));
        System.out.println("-3/b = "+Fraction.quot(-3, b));
        System.out.println("4/12 = "+Fraction.quot(4, 12));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        try {
            Fraction c = new Fraction(0, 5);
            System.out.println("c = "+c);
            System.out.println("a/c = "+Fraction.quot(a, c));
        } catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
        try {
            System.out.println("a/0 = "+Fraction.quot(a, 0));
        } catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Zadanie 20.11. Z20_11.java, Fraction.java

Dołączmy do klasy `Fraction` prywatną metodę obliczającą najmniejszą wspólną wielokrotność pary liczb dodatnich (w metodzie nie sprawdzamy, czy ten warunek jest spełniony, ponieważ gwarantuje nam to przyjęte założenie, że w obiekcie mianowniki są liczbami dodatnimi).

```
/** Zwraca najmniejszą wspólną wielokrotność pary liczb całkowitych
 * dodatnich <code>m</code> i <code>n</code> podanych jako parametry
 * wywołania metody.
 * @param m liczba całkowita dodatnia,
 * @param n liczba całkowita dodatnia.
 */
    private int nww(int m, int n) {
        int wm = m;
        int wn = n;
        while (wm != wn)
            if (wm > wn)
                wn += n;
            else
                wm += m;
        return wm;
    }
```

Czytelnik może zastosować inny algorytm wyznaczania NWW.

Porównujemy mianowniki obu ułamków. Jeśli są równe, to ułamki dodajemy według wzoru $\frac{a}{b}+\frac{c}{b}=\frac{a+c}{b}$ i zwracamy wynik (`new Fraction(this.num+x.num, this.den)`). Gdy mianowniki są różne, to obliczamy najmniejszą wspólną wielokrotność mianowników (`int wm = nww(this.den, x.den)`), rozszerzamy ułamki do wspólnego mianownika (`wm`) i następnie je dodajemy — $\frac{a}{b}+\frac{c}{d}=\frac{\frac{wm}{b}a}{\frac{wm}{b}b}+\frac{\frac{wm}{d}c}{\frac{wm}{d}d}=\frac{\frac{wm}{b}a}{wm}+\frac{\frac{wm}{d}c}{wm}=\frac{\frac{wm}{b}a+\frac{wm}{d}c}{wm}$.

```
/** Zwraca nowy obiekt <code>Fraction</code> - sumę obiektu wywołującego
 * tę metodę i obiektu przekazanego jako parametr.
 * @param x ułamek, obiekt klasy <code>Fraction</code>.
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public Fraction add(Fraction x) {
        if (this.den == x.den)
            return new Fraction(this.num+x.num, this.den).reduce();
        else {
            int wm = nww(this.den, x.den);
            return new Fraction(wm/this.den*this.num+wm/x.den*x.num, wm).
                reduce();
        }
    }
```

Dodawanie liczby całkowitej do ułamka zrealizujemy według wzoru

$$\frac{a}{b}+m=\frac{a}{b}+\frac{mb}{b}=\frac{a+mb}{b}.$$

```
/** Zwraca nowy obiekt <code>Fraction</code> - sumę obiektu wywołującego
 * tę metodę i liczby całkowitej przekazanej jako parametr.
 * @param m liczba całkowita typu <code>int</code>.
 */
    public Fraction add(int m) {
        return new Fraction(this.num+m*this.den, this.den).reduce();
    }
```

Pozostaje jeszcze przetestowanie działania metody `add()`.

```
/** <h3>Zadanie Z20.11</h3>
 * Tworzenie metod <code>add(Fraction)</code> i <code>add(int)</code>,
 * zwracających nowy obiekt będący sumą ułamka reprezentowanego
 * przez obiekt i drugiego ułamka lub liczby całkowitej.
 */
public class Z20_11 {
/** Prezentuje działanie metod zwracających obiekt reprezentujący sumę
 * ułamków i sumę ułamka i liczby całkowitej.
 */
    public static void main(String[] args) {
        System.out.println("3/4 + 1/2 = "+new Fraction(3, 4).
            add(new Fraction(1, 2)));
        Fraction a = new Fraction(-5, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a+b = "+a.add(b));
        System.out.println("b+a = "+b.add(a));
        System.out.println("a+2 = "+a.add(2));
        System.out.println("a-3 = "+a.add(-3));
    }
}
```

Zadanie 20.12. Z20_12.java, Fraction.java

Sumę dwóch obiektów `x` i `y` obliczymy, wykorzystując metodę `add()` dla jednego obiektu, a drugi podamy jako parametr (`x.add(y)`).

```
/** Zwraca obiekt klasy <code>Fraction</code> będący sumą dwóch
 * obiektów <code>Fraction</code>.
 * @param x obiekt klasy <code>Fraction</code>,
 * @param y obiekt klasy <code>Fraction</code>.
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static Fraction sum(Fraction x, Fraction y) {
        return x.add(y);
    }
```

Podobnie obliczymy sumę obiektu `x` i liczby całkowitej `m` (`x.add(m)`) oraz — korzystając z przemienności dodawania — sumę liczby całkowitej `m` i obiektu `x`.

```
/** Zwraca obiekt klasy <code>Fraction</code> będący sumą obiektu
 * <code>Fraction</code> i liczby całkowitej typu <code>int</code>.
 * @param x obiekt klasy <code>Fraction</code>,
 * @param m liczba całkowita typu <code>int</code>.
 */
    public static Fraction sum(Fraction x, int m) {
        return x.add(m);
    }

/** Zwraca obiekt klasy <code>Fraction</code> będący sumą liczby
 * całkowitej typu <code>int</code> i obiektu <code>Fraction</code>.
 * @param m liczba całkowita typu <code>int</code>,
 * @param x obiekt klasy <code>Fraction</code>.
 */
    public static Fraction sum(int m, Fraction x) {
        return x.add(m);
    }
```

Do kompletu metod (`sum()`) dołączymy metodę dodającą dwie liczby całkowite i zwracającą wynik w postaci obiektu klasy `Fraction`.

```
/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący sumę
 * dwóch liczb całkowitych typu <code>int</code>.
 * @param m liczba całkowita typu <code>int</code>,
 * @param n liczba całkowita typu <code>int</code>.
 */
    public static Fraction sum(int m, int n) {
        return new Fraction(m+n);
    }
```

Przykłady kilku dodawań ilustrują zastosowanie statycznych metod wykonujących dodawanie.

```
/** <h3>Zadanie Z20.12</h3>
 * Tworzenie metod statycznych realizujących dodawanie dwóch obiektów
 * <code>Fraction</code> oraz obiektu <code>Fraction</code>
 * i liczby całkowitej.
 */
public class Z20_12 {
/** Prezentuje działanie metod statycznych obliczających sumę
 * dwóch ułamków oraz ułamka i liczby całkowitej.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a+b = "+Fraction.sum(a, b));
        System.out.println("a+4 = "+Fraction.sum(a, 4));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("-3+b = "+Fraction.sum(-3, b));
        System.out.println("2+3 = "+Fraction.sum(2, 3));
    }
}
```

Zadanie 20.13. Z20_13.java, Fraction.java

Zamiana ułamka na przeciwny polega na zmianie znaku licznika (przyjęliśmy zasadę, że mianownik jest zawsze dodatni).

```
/** Zwraca obiekt <code>Fraction</code> reprezentujący ułamek przeciwny
 * do ułamka zawartego w obiekcie wywołującym tę metodę.
 */
    public Fraction addInv() {
        return new Fraction(-this.num, this.den);
    }

/** Zwraca obiekt <code>Fraction</code> reprezentujący ułamek przeciwny
 * do ułamka zawartego w obiekcie <code>x</code> podanym jako parametr.
 * @param x obiekt klasy <code>Fraction</code>.
 */
    public static Fraction addInv(Fraction x) {
        return new Fraction(-x.num, x.den);
    }
```

Pokazując działanie metody wyznaczającej ułamek przeciwny, wykonamy dodawanie dwóch liczb (ułamków) wzajemnie przeciwnych (wynik $^{0}/_{1}$) oraz odejmowanie ułamków w myśl szkolnej reguły mówiącej, że *odejmowanie można zastąpić dodawaniem liczby przeciwnej*.

```
/** <h3>Zadanie Z20.13</h3>
 * Tworzenie metod wyznaczających ułamek przeciwny.
 */
public class Z20_13 {
/** Prezentuje działanie metod obliczających ułamek przeciwny.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        System.out.println("-a = "+a.addInv());
        Fraction b = new Fraction(-7, 9);
        System.out.println("b = "+b);
        System.out.println("-b = "+Fraction.addInv(b));
        System.out.println("a+(-a) = "+a.add(a.addInv()));
        System.out.println("(-a)+a = "+a.addInv().add(a));
        System.out.println("a-b = "+a.add(b.addInv()));
        System.out.println("a-b = "+b.addInv().add(a));
        System.out.println("a-b = "+Fraction.sum(a,Fraction.addInv(b)));
     }
}
```

Zadanie 20.14. Z20_14.java, Fraction.java

W rozwiązaniu zadania 20.11 dokładnie omówiono dodawanie ułamków. Odejmowanie realizujemy w podobny sposób — we wzorach symbol dodawania (+) zastąpimy symbolem odejmowania (-). Nie zapomnijmy zmienić nazwy metody.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zwraca nowy obiekt <code>Fraction</code> - różnicę obiektu
 * wywołującego tę metodę i obiektu przekazanego jako parametr.
 * @param x ułamek, obiekt klasy <code>Fraction</code>.
 */
    public Fraction sub(Fraction x) {
        if (this.den == x.den)
            return new Fraction(this.num-x.num, this.den).reduce();
        else {
            int wm = nww(this.den, x.den);
            return new Fraction(wm/this.den*this.num-wm/x.den*x.num, wm).
                reduce();
        }
    }

/** Zwraca nowy obiekt <code>Fraction</code> - różnicę obiektu
 * wywołującego tę metodę (odjemna) i liczby całkowitej przekazanej jako
 * parametr (odjemnik).
 * @param m liczba całkowita typu <code>int</code> (odjemnik).
 */
    public Fraction sub(int m) {
        return new Fraction(this.num-m*this.den, this.den).reduce();
    }
```

Metodę `sub()` sprawdzimy, wykonując kilka przykładowych działań.

```
/** <h3>Zadanie Z20.14</h3>
 * Tworzenie metod <code>sub(Fraction)</code> i <code>sub(int)</code>
 * zwracających nowy obiekt będący różnicą ułamka reprezentowanego przez
 * obiekt i drugiego ułamka lub liczby całkowitej.
 */
public class Z20_14 {
/** Prezentuje działanie metod zwracających obiekt reprezentujący różnicę
 * ułamków i różnicę ułamka i liczby całkowitej.
 */
    public static void main(String[] args) {
        System.out.println("3/4 - 1/2 = "+new Fraction(3, 4).
            sub(new Fraction(1, 2)));
        Fraction a = new Fraction(-5, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a-b = "+a.sub(b));
        System.out.println("b-a = "+b.sub(a));
        System.out.println("a-2 = "+a.sub(2));
        System.out.println("a-(-3) = "+a.sub(-3));
    }
}
```

Zadanie 20.15. Z20_15.java, Fraction.java

Różnicę dwóch obiektów `x` i `y` obliczymy, wykorzystując metodę `sub()` dla pierwszego obiektu, a drugi podamy jako parametr (`x.sub(y)`).

```
/** Zwraca obiekt klasy <code>Fraction</code> będący różnicą dwóch
 * obiektów <code>Fraction</code>.
 * @param x obiekt klasy <code>Fraction</code> (odjemna),
 * @param y obiekt klasy <code>Fraction</code> (odjemnik).
 */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static Fraction diff(Fraction x, Fraction y) {
        return x.sub(y);
    }
```

Podobnie od obiektu `x` odejmiemy liczbę całkowitą `m` (`x.sub(m)`).

```
/** Zwraca obiekt klasy <code>Fraction</code> będący różnicą obiektu
 * <code>Fraction</code> (odjemna) i liczby całkowitej typu
 * <code>int</code> (odjemnik).
 * @param x obiekt klasy <code>Fraction</code> (odjemna),
 * @param m liczba całkowita typu <code>int</code> (odjemnik).
 */
    public static Fraction diff(Fraction x, int m) {
        return x.sub(m);
    }
```

Odejmowanie nie jest przemienne, ale obowiązuje wzór $m - x = -(x - m)$. Prawą stronę tego wzoru możemy zapisać, używając odpowiednich metod `x.sub(m).addInv()`.

```
/** Zwraca obiekt klasy <code>Fraction</code> będący różnicą liczby
 * całkowitej typu <code>int</code> (odjemna) i obiektu
 * <code>Fraction</code> (odjemnik).
 * @param m liczba całkowita typu <code>int</code> (odjemna),
 * @param x obiekt klasy <code>Fraction</code> (odjemnik).
 */
    public static Fraction diff(int m, Fraction x) {
        return x.sub(m).addInv();
    }
```

Kolejna metoda odejmuje dwie liczby całkowite i zwraca wynik w postaci obiektu `Fraction`.

```
/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący różnicę
 * dwóch liczb całkowitych typu <code>int</code>.
 * @param m liczba całkowita typu <code>int</code> (odjemna),
 * @param n liczba całkowita typu <code>int</code> (odjemnik).
 */
    public static Fraction diff(int m, int n) {
        return new Fraction(m-n);
    }
```

Kilka przykładowych obliczeń pokaże nam działanie metody `diff()`.

```
/** <h3>Zadanie Z20.15</h3>
 * Tworzenie metod statycznych realizujących odejmowanie dwóch obiektów
 * <code>Fraction</code>, obiektu <code>Fraction</code>
 * i liczby całkowitej.
 */
public class Z20_15 {
/** Prezentuje działanie metod statycznych obliczających różnicę dwóch
 * ułamków oraz ułamka i liczby całkowitej.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("a = "+a);
        Fraction b = new Fraction(7, 15);
        System.out.println("b = "+b);
        System.out.println("a-b = "+Fraction.diff(a, b));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("b-a = "+Fraction.diff(b, a));
        System.out.println("a-4 = "+Fraction.diff(a, 4));
        System.out.println("(-3)-b = "+Fraction.diff(-3, b));
        System.out.println("b-(-3) = "+Fraction.diff(b, -3));
        System.out.println("2-3 = "+Fraction.diff(2, 3));
    }
}
```

Zadanie 20.16. Z20_16.java, Fraction.java

Wartość prywatnych pól obiektu (licznik i mianownik ułamka) możemy odczytać przy użyciu metod `getNum()` i `getDen()` (czasem taka informacja może być przydatna).

```
/** Zwraca wartość licznika ułamka reprezentowanego przez obiekt.
 */
    public int getNum() {
        return this.num;
    }

/** Zwraca wartość mianownika ułamka reprezentowanego przez obiekt.
 */
    public int getDen() {
        return this.den;
    }
```

Przykład odczytywania prywatnych pól obiektu z zastosowaniem zdefiniowanych metod.

```
/** <h3>Zadanie Z20.16</h3>
 * Tworzenie metod pobierających dane z prywatnych pól obiektu
 * <code>Fraction</code>.
 */
public class Z20_16 {
/** Prezentuje działanie metod pobierających dane z prywatnych pól
 * obiektu <code>Fraction</code> - licznik i mianownik ułamka.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, -7);
        System.out.println("a = "+a);
        System.out.println("licznik: "+a.getNum());
        System.out.println("mianownik: "+a.getDen());
    }
}
```

Zadanie 20.17. Z20_17.java, Fraction.java

Czasem potrzebujemy zmienić wartość wybranych (prywatnych) pól obiektu bez tworzenia nowego obiektu i usuwania poprzedniej instancji. Możemy to zrealizować, definiując odpowiednie metody (szczegóły w komentarzach).

```
/** Ustawia wartość licznika ułamka reprezentowanego przez obiekt.
 * @param m liczba całkowita typu <code>int</code>, nowa wartość
 * licznika ułamka reprezentowanego przez obiekt.
 */
    public Fraction setNum(int m) {
        this.num = m;
        return this;
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Ustawia wartość mianownika ułamka.
 * @param  n liczba całkowita dodatnia typu <code>int</code>, nowa
 * wartość mianownika ułamka reprezentowanego przez obiekt.
 * @throws IllegalArgumentException, gdy podamy mianownik ujemny lub
 * równy 0.
 */
   public Fraction setDen(int n) {
       if (n > 0) {
           this.den = n;
           return this;
       } else
           throw new IllegalArgumentException("Parametr n <= 0!");
   }

/** Ustawia wartość licznika i mianownika ułamka.
 * @param m liczba całkowita typu <code>int</code>, nowa wartość
 * licznika ułamka reprezentowanego przez obiekt.
 * @param  n liczba całkowita dodatnia typu <code>int</code>, nowa
 * wartość mianownika ułamka reprezentowanego przez obiekt.
 * @throws IllegalArgumentException, gdy podamy mianownik ujemny lub
 * równy 0.
 */
   public Fraction setFrac(int m, int n) {
       if (n > 0) {
           this.num = m;
           this.den = n;
           return this;
       } else
           throw new IllegalArgumentException("Parametr n <= 0!");
   }
```

Przykład wykorzystania metod `setNum()`, `setDen()` i `setFrac()` pokazuje poprawne zastosowanie metod oraz sytuacje prowadzące do błędu (z obsługą wyjątków zgłaszanych przez metody — próba ustawienia mianownika ujemnego lub wartości *0*).

```
/** <h3>Zadanie Z20.17</h3>
 * Tworzenie metod zmieniających dane w prywatnych polach obiektu
 * <code>Fraction</code>.
 */
public class Z20_17 {
/** Prezentuje działanie metod zmieniających dane w prywatnych polach
 * obiektu <code>Fraction</code>.
 */
   public static void main(String[] args) {
       Fraction a = new Fraction(4, -7);
       System.out.println("a = "+a);
       System.out.println("Zmiana wartości:");
       System.out.println(" - licznika (3), a = "+a.setNum(3));
       System.out.println(" - mianownika (11), a = "+a.setDen(11));
       System.out.println(" - ułamka (3, 5), a = "+a.setFrac(3, 5));
       try {
           a.setDen(-2);
       } catch (IllegalArgumentException e) {
           System.out.println("Mianownik: "+e.getMessage());
       }
       try {
           a.setFrac(-2, 0);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            } catch (IllegalArgumentException e) {
                System.out.println("Mianownik: "+e.getMessage());
            }
        }
    }
```

Zadanie 20.18. Z20_18.java, Fraction.java

Adnotacja `Override` informuje kompilator, że metoda `Fraction.equals()` przesłania dziedziczoną metodę `Object.equals()`. Parametr `o` jest obiektem klasy `Object`, z której dziedziczą wszystkie klasy. Najpierw referencję obiektu wywołującego metodę `equals` porównujemy z referencją parametru (`this == o`) — równość referencji gwarantuje równość obiektów (obie referencje wskazują ten sam obiekt) i możemy zwrócić wynik `true`.

Jeśli porównanie nie zakończyło się, to sprawdzamy, czy parametr `o` jest referencją do jakiegokolwiek obiektu. Spełnienie równości `o == null` spowoduje zwrócenie wartości `false` (obiekt wywołujący metodę ma referencję różną od `null`). Wartość `false` zwrócimy również wtedy, gdy parametr `o` jest obiektem innej klasy niż `Fraction`, czyli nie jest obiektem tej klasy (`!(o instanceof Fraction)`).

Jeśli porównywanie trwa nadal, to mamy już do czynienia z dwoma obiektami klasy `Fraction`. Tworzymy nowy obiekt klasy `Fraction` i rzutujemy parametr `o` klasy `Object` na typ obiektowy `Fraction` (`Fraction c = (Fraction) o`). Teraz pozostaje porównanie dwóch obiektów klasy `Fraction` — obiektu wywołującego metodę i obiektu `c`:

```
if (this.num*c.den == this.den*c.num)
    return true;
else
    return false;
```

Jak wspomniano w uwadze do zadania, podczas obliczania iloczynów w porównaniu (`this.num*c.den == this.den*c.num`) możemy przekroczyć maksymalną wartość typu `int`, więc możemy otrzymać błędny wynik.

```
/** Porównuje obiekt klasy <code>Fraction</code> (wywołujący tę metodę)
 * z dowolnym innym obiektem i zwraca wartość <code>true</code>, gdy
 * obiekty są tego samego typu i reprezentują tę samą wartość (ułamek).
 * Metoda przesłania metodę <code>Object.equals()</code>.
 */
   @Override public boolean equals(Object o) {
      if (this == o)
          return true;
      if (o == null || !(o instanceof Fraction))
          return false;
      Fraction c = (Fraction) o;
      if (this.num*c.den == this.den*c.num)
          return true;
      else
          return false;
   }
```

Tworząc kopię (obiekt `d`) obiektu wywołującego metodę, możemy obiekty `c` i `d` skrócić. Porównanie skróconych ułamków ograniczy się do porównania ich liczników

Ebookpoint.pl kopia dla: pawher@o2.pl

i mianowników (bez mnożenia). Kopię obiektu robimy po to, aby w wyniku skracania nie zmienić pól obiektu wywołującego metodę.

```
Fraction d = new Fraction(this);
c.reduce();
d.reduce();
if (d.num == c.num && d.den == c.den)
    return true;
else
    return false;
```


Uwaga

Przedefiniowanie metody `hashCode()` nie jest w tym zadaniu konieczne. Pokażemy jednak, jak można tę metodę zrealizować[1]:

```
@Override public int hashCode() {
    int d = nwd(num, den);
    int result = 17;
    result = 31*result+num/d;
    result = 31*result+den/d;
    return result;
}
```

Analizę kodu przykładu zastosowania metody `equals()` dla obiektów klasy `Fraction` pozostawiamy Czytelnikowi.

```
/** <h3>Zadanie Z20.18</h3>
 * Tworzenie metody equals() do porównywania obiektu <code>Fraction</code>
 * z innymi obiektami tej samej klasy lub innych klas.
 */
public class Z20_18 {
/** Prezentuje działanie metody equals() przesłaniającej metodę
 * Object.equals.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        Fraction b = new Fraction(12, 21);
        Fraction c = new Fraction(20, 35);
        Fraction d = new Fraction(20, 36);
        System.out.println("Trzy różne obiekty reprezentują tę samą
           liczbę 4/7.");
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("c = "+c);
        System.out.println("Czwarty obiekt reprezentuje inną liczbę
            20/36 = 5/9.");
        System.out.println("d = "+d);
        System.out.println("Metoda equals() jest relacją równoważności:");
        System.out.println(" * jest zwrotna:");
        System.out.println("a.equals(a): "+a.equals(a));
        System.out.println(" * jest symetryczna:");
        System.out.println("a.equals(b): "+a.equals(b));
        System.out.println("b.equals(a): "+b.equals(a));
        System.out.println();
```

[1] Opracowane na podstawie: J. Bloch, *Java. Efektywne programowanie. Wydanie II*, Helion 2009.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("a.equals(d): "+a.equals(d));
        System.out.println("d.equals(a): "+d.equals(a));
        System.out.println(" * jest przechodnia:");
        System.out.println("a.equals(b): "+a.equals(b));
        System.out.println("b.equals(c): "+b.equals(c));
        System.out.println("a.equals(c): "+a.equals(c));
        System.out.println("Podstawiamy d = a, obie referencje wskazują
            ten sam obiekt.");
        d = a;
        System.out.println("a.equals(d): "+a.equals(d));
        System.out.println("d.equals(a): "+d.equals(a));
        System.out.println("Porównujemy obiekty różnych typów:");
        System.out.println("a.equals(5): "+a.equals(5));
        System.out.println("Zmieniamy wartość obiektu a. Referencja d
            nadal wskazuje ten sam obiekt.");
        a.setFrac(5, 1);
        System.out.println("a = "+a);
        System.out.println("d = "+d);
        System.out.println("a.equals(d): "+a.equals(d));
        System.out.println("d.equals(a): "+d.equals(a));
        System.out.println("Porównujemy obiekty różnych typów:");
        System.out.println("a.equals(Integer.valueOf(5)): "
            +a.equals(Integer.valueOf(5)));
        System.out.println("Podstawiamy d = null, referencja d nie
            wskazuje żadnego obiektu.");
        d = null;
        System.out.println("a.equals(d): "+a.equals(d));
        System.out.println("b.equals(null): "+b.equals(null));
    }
}
```

Zadanie 20.19. Z20_19.java, Fraction.java

Rzutujemy licznik ułamka (`this.num`) na typ zmiennoprzecinkowy `double` lub `float` i dzielimy przez mianownik (`this.den`). W wyniku dzielenia otrzymujemy liczbę zmiennoprzecinkową odpowiedniego typu.

```
/** Zwraca wartość dziesiętną typu <code>double</code> ułamka
 * reprezentowanego przez obiekt wywołujący tę metodę.
 */
    public double doubleValue() {
        return (double)this.num/this.den;
    }

/** Zwraca wartość dziesiętną typu <code>float</code> ułamka
 * reprezentowanego przez obiekt wywołujący tę metodę.
 */
    public float floatValue() {
        return (float)this.num/this.den;
    }
```

Podobnie postępujemy w metodach statycznych, dzieląc licznik (rzutowany na typ zmiennoprzecinkowy) przez mianownik.

```
/** Zwraca wartość dziesiętną typu <code>double</code> ułamka
 * reprezentowanego przez obiekt podany jako parametr.
 * @param x obiekt typu <code>Fraction</code>.
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
 */
    public static double toDouble(Fraction x) {
        return (double)x.num/x.den;
    }

/** Zwraca wartość dziesiętną typu <code>float</code> ułamka
 * reprezentowanego przez obiekt podany jako parametr.
 * @param x obiekt typu <code>Fraction</code>.
 */
    public static float toFloat(Fraction x) {
        return (float)x.num/x.den;
    }
```

Działanie metod ilustrujemy czterema przykładami obliczeń:

```
/** <h3>Zadanie Z20.19</h3>
 * Tworzenie metod zamieniających ułamek zwykły (obiekt
 * <code>Fraction</code>) na ułamek dziesiętny.
 */
public class Z20_19 {
/** Prezentuje działanie metod zamieniających ułamek zwykły (obiekt
 * <code>Fraction</code>) na ułamek dziesiętny.
 */
    public static void main(String[] args) {
        Fraction a = new Fraction(4, 7);
        System.out.println("Double: a = "+a+" = "+a.doubleValue());
        System.out.println("Float:  a = "+a+" = "+a.floatValue());
        Fraction b = new Fraction(17, 25);
        double x = Fraction.toDouble(b);
        System.out.println("Double: b = "+b+" = "+x);
        System.out.println("Float:  b = "+b+" = "+Fraction.toFloat(b));
    }
}
```

Zadanie 20.20. Z20_20.java, Fraction.java

Mamy cztery poprawne postacie łańcucha wejściowego `str` — liczbę całkowitą lub dwie liczby całkowite oddzielone znakiem *slash* (`/`), *kropką* (`.`) albo przecinkiem (`,`).

a) Jeśli w łańcuchu wejściowym `str` nie występuje żaden z tych znaków, to mamy przypuszczalnie do czynienia z liczbą całkowitą (np. `"23"`). Polom obiektu przypisujemy wartości:

```
this.num = Integer.parseInt(str);
this.den = 1;
```

Gdy łańcuch `str` nie jest poprawnym zapisem liczby całkowitej, to użyta do konwersji metoda `Integer.parseInt()` rzuci wyjątek.

b) Znajdujący się na pozycji `i = str.indexOf('/')` znak *slash* (`/`) rozdziela łańcuch znaków na dwie liczby — licznik (`str.substring(0, i)`) i mianownik (`str.substring(i+1)`). Polom obiektu przypisujemy wartości:

```
this.num = Integer.parseInt(str.substring(0, i));
this.den = Integer.parseInt(str.substring(i+1));
```

Jeśli mianownik okaże się zerem, to rzucimy wyjątek:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
if (this.den == 0)
    throw new ArithmeticException("Dzielenie przez 0!");
```

Gdy jeden z podciągów (`str.substring(0, i)` lub `str.substring(i+1)`) nie jest poprawną liczbą całkowitą, to metoda `Integer.parseInt()` rzuci wyjątek.

c) Występująca na pozycji `i = str.indexOf('.')` kropka (`.`) rozdziela łańcuch na dwie części — część całkowitą liczby (`str.substring(0, i)`) i ciąg cyfr części ułamkowej (`str.substring(i+1)`). Rozważmy to na przykładzie:

$$"2.75" \rightarrow "2" \; i \; "75" \rightarrow 2\frac{75}{100} = \frac{275}{100}.$$

Suma (złączenie) tych dwóch podciągów daje nam licznik ułamka:

```
this.num = Integer.parseInt(str.substring(0, i)+str.substring(i+1));
```

Drugi podciąg wyznacza mianownik ułamka (potęga liczby 10) — liczba cyfr jest równa liczbie zer w mianowniku.

```
int k = str.length()-i; // liczba cyfr po kropce
this.den = 1;
int n = 1;
while (n++ < k)
    this.den *= 10;
```

Błąd w postaci liczby może być sygnalizowany wyjątkami zgłoszonymi przez metodę `parseInt()`.

d) Przecinek (`,`) rozdziela łańcuch — według przykładu z podpunktu c) (kod jest wspólny dla obu podpunktów).

Na koniec zbudowany ułamek podlega korekcji (tak aby mianownik był dodatni) oraz skróceniu.

```
/** Tworzy nowy obiekt <code>Fraction</code> - ułamek na
 * podstawie łańcucha znaków o postaci "4/7", "5", "2.45" lub "2,45".
 * @param str łańcuch znaków o postaci "4/7", "5", "2.45" lub "2,45".
 * @throws NumberFormatException, gdy podamy łańcuch znaków nie
 * przedstawia ułamka zwykłego lub dziesiętnego.
 */
   public Fraction(String str) {
       int i = str.indexOf('.');
       if (i == -1)
           i = str.indexOf(',');
       if (i == -1) {
           i = str.indexOf('/');
           if (i == -1) {
               this.num = Integer.parseInt(str);
               this.den = 1;
           } else {
               this.num = Integer.parseInt(str.substring(0, i));
               this.den = Integer.parseInt(str.substring(i+1));
               if (this.den == 0)
                   throw new ArithmeticException("Dzielenie przez 0!");
           }
       } else {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            this.num = Integer.parseInt(str.substring(0, i)+
                str.substring(i+1));
            int k = str.length()-i;
            this.den = 1;
            int n = 1;
            while (n++ < k)
                this.den *= 10;
        }
        this.correction();
        this.reduce();
    }
```

W przykładzie pokazujemy działanie zdefiniowanego konstruktora tworzącego obiekt na podstawie różnych łańcuchów znaków.

```
/** <h3>Zadanie Z20.20</h3>
 * Tworzenie konstruktora budującego obiekt <code>Fraction</code>) na
 * podstawie łańcucha znaków o postaci "4/7", "5", "2.45" lub "2,45".
 */
public class Z20_20 {
/** Prezentuje działanie konstruktora <code>Fraction(String)</code>).
 */
    public static void main(String[] args) {
        Fraction a = new Fraction("14/35");
        System.out.println("a = "+a);
        Fraction b = new Fraction("3");
        System.out.println("b = "+b);
        Fraction c = new Fraction("2.45");
        System.out.println("c = "+c);
        Fraction d = new Fraction("2,45");
        System.out.println("d = "+d);
        try {
            Fraction e = new Fraction("45;35");
            System.out.println("e = "+e);
        } catch (NumberFormatException e) {
            System.out.println("Niewłaściwy parametr - "+e.getMessage());
        }
    }
}
```

Zadanie 20.21. Z20_21.java, Fraction.java

Liczbę zmiennoprzecinkową (typu `float` lub `double`) zamieniamy na łańcuch znaków, wykorzystując operator konkatenacji (+), czyli łączenia tekstów, oraz konstruktor zbudowany w rozwiązaniu zadania 20.20 (`new Fraction(""+x)`). W tej operacji `""+x` liczba `x` zamieniana jest na łańcuch znaków i łączona z pustym łańcuchem.

```
/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący określoną
 * wartość typu <code>float</code>.
 * @param x liczba zmiennoprzecinkowa typu <code>float</code>.
 */
    public static Fraction valueOf(float x) {
        return new Fraction(""+x);
    }

/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący określoną
 * wartość typu <code>double</code>.
 * @param x liczba zmiennoprzecinkowa typu <code>double</code>.
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
 */
    public static Fraction valueOf(double x) {
        return new Fraction(""+x);
    }
```

Kolejne metody statyczne zwracają obiekt klasy `Fraction` odpowiadający liczbie całkowitej `m` (`m/1`) lub parze liczb całkowitych `m` i `n` (`m/n`).

```
/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący określoną
 * wartość typu całkowitego <code>int</code>.
 * @param m liczba całkowita.
 */
    public static Fraction valueOf(int m) {
        return new Fraction(m);
    }

/** Zwraca obiekt klasy <code>Fraction</code> reprezentujący ułamek
 * <code>m/n</code> na podstawie wartości całkowitych parametrów
 * <code>m</code> i <code>n</code>.
 * @param m liczba całkowita (licznik ułamka),
 * @param n liczba całkowita (mianownik ułamka).
 * @throws IllegalArgumentException, gdy podamy mianownik n = 0.
 */
    public static Fraction valueOf(int m, int n) {
        if (n != 0)
            return new Fraction(m, n);
        else
            throw new IllegalArgumentException("Parametr n = 0!");
    }
```

Ostatnia z omawianych metod zamienia łańcuch znaków (o odpowiedniej postaci) na obiekt `Fraction`.

```
/** Zwraca obiekt klasy <code>Fraction</code> na podstawie łańcucha
 * znaków przedstawiającego ułamek dziesiętny lub zwykły.
 * @param str łańcuch znaków przedstawiający ułamek dziesiętny "0.12",
 * "0,12" lub zwykły "2/3".
 * @throws NumberFormatException wystąpi wyjątek, gdy parametr
 * <code>str</code> nie jest liczbą całkowitą lub ułamkiem dziesiętnym
 * (zapis w notacji naukowej nie jest interpretowany).
 */
    public static Fraction valueOf(String str) {
        return new Fraction(str);
    }
```

W przykładzie pokazujemy działanie zdefiniowanych metod statycznych tworzących obiekt na podstawie parametrów różnych typów.

```
/** <h3>Zadanie Z20.21</h3>
 * Tworzenie metod statycznych <code>valueOf()</code> zwracających
 * obiekt <code>Fraction</code>, zbudowanych na podstawie parametrów
 * różnych typów.
 */
public class Z20_21 {
/** Prezentuje działanie metod statycznych <code>valueOf()</code>.
 */
    public static void main(String[] args) {
        System.out.println("a = "+Fraction.valueOf("2.45"));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("b = "+Fraction.valueOf("2,45"));
        System.out.println("c = "+Fraction.valueOf("15/45"));
        System.out.println("d = "+Fraction.valueOf(15, 45));
        System.out.println("e = "+Fraction.valueOf(5));
        try {
            Fraction e = Fraction.valueOf("45;35");
        } catch (NumberFormatException e) {
            System.out.println("Niewłaściwy parametr - "+e.getMessage());
        }
        try {
            Fraction e = Fraction.valueOf("15/0");
        } catch (ArithmeticException e) {
            System.out.println("Niewłaściwy parametr - "+e.getMessage());
        }
    }
}
```

Zadanie 20.22. Z20_22.java

Tworzymy obiekt `a` o wartości początkowej `1` (`Fraction a = new Fraction(1)`). Zauważmy, że kolejne wyrazy ciągu spełniają zależność $a_{n+1} = 1 + \frac{1}{a_n}$, co możemy zapisać w postaci instrukcji podstawiania `a = a.multInv().add(1)`. Obliczenia kolejnych wyrazów wykonamy w pętli.

```
/** <h3>Zadanie 20.22</h3> */
public class Z20_22 {
    public static void main(String[] args) {
        Fraction a = new Fraction(1);
        for(int i = 1; i <= 15; ++i) {
            System.out.println("a("+i+") = "+a+" = "+a.doubleValue());
            a = a.multInv().add(1);
        }
    }
}
```

Maksymalnie możemy obliczyć 45 wyrazów ciągu.


Uwaga

Możemy zauważyć, że liczniki kolejnych wyrazów ciągu (dotyczy to również mianowników) są liczbami Fibonacciego. Ciąg jest zbieżny do liczby φ (stosunek złotego podziału odcinka).

Zadanie 20.23. Z20_23.java

Zauważmy, że każdy następny składnik sumy $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots$ jest 2 razy mniejszy od poprzedniego. Ustawiamy wartości początkowe `a = 1; suma = a;` (`Fraction a = new Fraction(1); Fraction suma = new Fraction(a);`). W pętli wyświetlamy wartość sumy (zmienna `suma`) i obliczamy następny składnik `a = a.div(2)` i kolejną wartość sumy `suma = suma.add(a)`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** <h3>Zadanie Z20.23</h3> */
public class Z20_23 {
    public static void main(String[] args) {
        Fraction a = new Fraction(1);
        Fraction suma = new Fraction(a);
        for(int i = 0; i < 30; ++i) {
            System.out.println("s("+i+") = "+suma+" = "+
                suma.doubleValue());
            a = a.div(2);
            suma = suma.add(a);
        }
    }
}
```

Możemy obliczyć co najwyżej 30 sum cząstkowych tego szeregu. Możemy zauważyć zbieżność sum cząstkowych do liczby 2. Istotnie jest to szereg geometryczny zbieżny $s = \frac{1}{1-\frac{1}{2}} = \frac{1}{\frac{1}{2}} = 2$.

Zadanie 20.24. Z20_24.java

Każdy następny składnik sumy $1-\frac{1}{2}+\frac{1}{4}-\frac{1}{8}+\frac{1}{16}-\frac{1}{32}+...$ jest 2 razy mniejszy od poprzedniego (dotyczy bezwzględnej wartości składników szeregu). Ustawiamy wartości początkowe `a = 1; suma = a;` (`Fraction a = new Fraction(1); Fraction suma = new Fraction(a);`). W pętli wyświetlamy wartość sumy (zmienna `suma`) i obliczamy następny składnik `a = a.div(2).addInv()` (zmieniamy znak wyrazu na przeciwny) i kolejną wartość sumy `suma = suma.add(a)`.

```
/** <h3>Zadanie Z20.24</h3> */
public class Z20_24 {
    public static void main(String[] args) {
        Fraction a = new Fraction(1);
        Fraction suma = new Fraction(a);
        for(int i = 1; i <= 30; ++i) {
            System.out.println("s("+i+") = "+suma+" = "+
                suma.doubleValue());
            a = a.div(2).addInv();
            suma = suma.add(a);
        }
    }
}
```

Zmianę znaków kolejnych składników możemy również zrealizować tak: `a = a.div(-2)`.

Możemy obliczyć co najwyżej 30 sum cząstkowych tego szeregu. Zwróćmy uwagę na zbieżność sum cząstkowych do liczby $\frac{2}{3} \approx 0{,}66666666666667$. Istotnie jest to szereg geometryczny zbieżny $s = \frac{1}{1-\left(-\frac{1}{2}\right)} = \frac{1}{\frac{3}{2}} = \frac{2}{3}$.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 20.25. Z20_25.java

Definiujemy pomocniczą stałą `ZERO` reprezentującą ułamek (obiekt `Fraction`) o wartości `0`. Współczynniki równania wczytujemy w postaci tekstu (używając skanera `input` i metody `next()`). Konstruktor `Fraction` zamieniający tekst na obiekt rozpoznaje poprawnie następujące postacie danych: `"2"`, `"2/3"`, `"2.4"` i `"2,4"` (również ze znakiem — na początku łańcucha).

Jeżeli współczynnik `a` nie jest zerem, to równanie ma jedno rozwiązanie: $x = -\frac{b}{a}$ (`b.div(a).addInv()`). Natomiast gdy `a` jest zerem, to wartość współczynnika `b` decyduje o tym, czy równanie jest tożsamościowe ($b = 0$), czy sprzeczne ($b \neq 0$).

```
import java.util.Scanner;
/** <h3>Zadanie Z20.25</h3> */
public class Z20_25 {
    private static final Fraction ZERO = new Fraction();
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Podaj współczynniki równania ax+b=0");
        System.out.print("a = ");
        Fraction a = new Fraction(input.next());
        System.out.print("b = ");
        Fraction b = new Fraction(input.next());
        if (!a.equals(ZERO))
            System.out.println("x = "+b.div(a).addInv());
        else if(b.equals(ZERO))
            System.out.println("Równanie tożsamościowe.");
        else
            System.out.println("Równanie sprzeczne.");
    }
}
```

Program można rozbudować, dodając powtarzanie obliczeń (w pętli) i obsługę wyjątków powstałych podczas wprowadzania danych.

Zadanie 20.26. Z20_26.java

Zgodnie z sugestią zawartą w podanej wskazówce definiujemy statyczną metodę `input Fraction()` do wprowadzania ułamków w postaci łańcucha znaków (np. `"5"`, `"3/4"` lub `"1.5"`). Dodatkowo definiujemy stałą `ZERO` typu `Fraction` reprezentującą ułamek `0/1`.

Do rozwiązania mamy układ dwóch równań z dwiema niewiadomymi: $\begin{cases} ax + by = e \\ cx + dy = f \end{cases}$.

Obliczamy główny wyznacznik układu $W = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - cb$ oraz wyznaczniki $W_x = \begin{vmatrix} e & b \\ f & d \end{vmatrix} = ed - fb$ (współczynniki przy zmiennej x zastępujemy wyrazami wolnymi) i $W_y = \begin{vmatrix} a & e \\ c & f \end{vmatrix} = af - ce$ (współczynniki przy zmiennej y zastępujemy wyrazami

Ebookpoint.pl kopia dla: pawher@o2.pl

wolnymi). Do obliczania tych wyznaczników budujemy metodę `det()` (ang. *determinant* — wyznacznik). Jeśli wyznacznik główny $W \neq 0$ (`!W.equals(ZERO)`), to rozwiązaniem układu jest para liczb $x = \frac{W_x}{W}$ i $y = \frac{W_y}{W}$. Gdy $W = W_x = W_y = 0$, to układ jest nieoznaczony (nieskończenie wiele par liczb x i y spełnia równania układu). Natomiast w przypadku $W = 0 \wedge \left(W_x \neq 0 \vee W_y \neq 0\right)$ układ jest sprzeczny.

```
import java.util.Scanner;
/** <h3>Zadanie Z20.26</h3> */
public class Z20_26 {
    private static final Fraction ZERO = new Fraction();
    private static Fraction inputFraction(String s) {
        Scanner input = new Scanner(System.in);
        System.out.print(s);
        return Fraction.valueOf(input.next());
    }
    private static Fraction det(Fraction a, Fraction b, Fraction c,
        Fraction d) {
        return Fraction.diff(Fraction.prod(a, d), Fraction.prod(c, b));
    }
    public static void main(String[] args) {
        System.out.println("Podaj współczynniki równania ax+by=e");
        Fraction a = inputFraction("a = ");
        Fraction b = inputFraction("b = ");
        Fraction e = inputFraction("e = ");
        System.out.println("Podaj współczynniki równania cx+dy=f");
        Fraction c = inputFraction("c = ");
        Fraction d = inputFraction("d = ");
        Fraction f = inputFraction("f = ");
        Fraction W  = det(a, b, c, d);
        Fraction Wx = det(e, b, f, d);
        Fraction Wy = det(a, e, c, f);
        if (!W.equals(ZERO)) {
            System.out.println("Układ oznaczony:");
            System.out.println("x = "+Wx.div(W));
            System.out.println("y = "+Wy.div(W));
        } else if(Wx.equals(ZERO) && Wy.equals(ZERO))
            System.out.println("Układ nieoznaczony.");
        else
            System.out.println("Układ sprzeczny.");
    }
}
```

Zadanie 20.27. Z20_27.java

Wzór iteracyjny $x = \frac{1}{2}\left(x + \frac{a}{x}\right)$ możemy zapisać w postaci `x = x.add(a.div(x)).div(2)` — mnożenie przez ½ zastępujemy dzieleniem przez *2*. Iterację powtarzamy czterokrotnie (pętla `for`), wyświetlając w każdym cyklu wartość obliczonego przybliżenia.

```
/** <h3>Zadanie Z20.27</h3> */
public class Z20_27 {
    public static void main(String[] args) {
        Fraction a = new Fraction(5);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        Fraction x = new Fraction(1);
        System.out.println("x = "+x+" = "+x.doubleValue());
        for(int i = 0; i < 4; ++i) {
            x = x.add(a.div(x)).div(2);
            System.out.println("x = "+x+" = "+x.doubleValue());
        }
        Fraction y = x.mult(x);
        System.out.println("Sprawdzenie x^2 = "+y+" = "+y.doubleValue());
    }
}
```

Otrzymany wynik sprawdzamy, podnosząc `x` do kwadratu (`x.mult(x)`).

21. Klasa opakowująca Angle — miara kąta i funkcje trygonometryczne

Zadanie 21.1. Z21_1.java, Angle.java

Objaśnienie poszczególnych metod zawarte jest w komentarzach dokumentacyjnych.

```
/** Klasa <code>Angle</code> jest klasą opakowującą. Zawiera jedno
 * prywatne pole typu <code>double</code>, przechowujące miarę kąta
 * wyrażonego w radianach.
 */
public class Angle {
/** Przechowuje miarę kąta reprezentowanego przez obiekt podaną w radianach.
 */
    private double x;

/** Tworzy nowy obiekt <code>Angle</code> reprezentujący kąt <code>x rad</code>.
 * @param x liczba typu <code>double</code>, miara kąta w radianach.
 */
    public Angle(double x) {
        this.x = x;
    }

/** Zwraca wartość funkcji sinus kąta reprezentowanego przez obiekt.
 */
    public double sin() { /* sinus */
        return Math.sin(this.x);
    }

/** Zwraca wartość funkcji cosinus kąta reprezentowanego przez obiekt.
 */
    public double cos() { /* cosinus */
        return Math.cos(this.x);
    }

/** Zwraca wartość funkcji tangens kąta reprezentowanego przez obiekt.
 */
    public double tan() { /* tangens */
        return Math.tan(this.x);
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    /** Zwraca wartość funkcji cotangens kąta reprezentowanego przez obiekt.
     */
        public double cot() { /* cotangens */
            return 1/Math.tan(this.x);
        }

    /** Zwraca wartość funkcji secans kąta reprezentowanego przez obiekt.
     */
        public double sec() { /* secans */
            return 1/Math.cos(this.x);
        }

    /** Zwraca wartość funkcji cosecans kąta reprezentowanego przez obiekt.
     */
        public double csc() { /* cosecans */
            return 1/Math.sin(this.x);
        }
    }
```

W podanym przykładzie najpierw tworzymy obiekt `alfa` reprezentujący kąt o mierze $\frac{\pi}{4} rad = 45°$. Następnie obliczamy wartości funkcji trygonometrycznych dla tego kąta.

```
public class Z21_1 {
    public static void main(String args[]) {
        Angle alfa = new Angle(Math.PI/4);
        System.out.println("sin(alfa) = "+alfa.sin());
        System.out.println("cos(alfa) = "+alfa.cos());
        System.out.println("tan(alfa) = "+alfa.tan());
        System.out.println("cot(alfa) = "+alfa.cot());
        System.out.println("sec(alfa) = "+alfa.sec());
        System.out.println("csc(alfa) = "+alfa.csc());
    }
}
```

Zadanie 21.2. Z21_2.java, Angle.java

Jedyne pole `x` obiektu klasy `Angle` przechowuje miarę kąta (reprezentowanego przez obiekt) wyrażoną w radianach i tę wielkość zwracamy jako wynik metody `radian()`.

```
    /** Zwraca miarę kąta reprezentowanego przez obiekt wyrażoną w radianach.
     */
        public double radian() {
            return x;
        }
```

Zamiany radianów (`x`) na stopnie dokonujemy według wzoru $\frac{x}{\pi} \cdot 180$ (`x/Math.PI*180`).

```
    /** Zwraca miarę kąta reprezentowanego przez obiekt wyrażoną w stopniach.
     */
        public double degree() {
            return x/Math.PI*180;
        }
```

Przykładowe obliczenia wykonamy dla kątów $\alpha = \frac{\pi}{4} rad$ i $\beta = 1\, rad$.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zadanie Z21.2 */
public class Z21_2 {
    public static void main(String args[]) {
        Angle alfa = new Angle(Math.PI/4);
        System.out.println("alfa = "+alfa.radian()+" rad");
        System.out.println("alfa = "+alfa.degree()+"\u00B0");
        Angle beta = new Angle(1.0);
        System.out.println("beta = "+beta.radian()+" rad");
        System.out.println("beta = "+beta.degree()+"\u00B0");
    }
}
```

Zadanie 21.3. Z21_3.java, Angle.java

Kąt pełny (jeden pełny obrót) dzielimy na *360* części i *1/360* część kąta pełnego nazywamy stopniem kątowym (*1°*). Stopień dzielimy na *60* części — *60* minut kątowych (*1′ = 1/60°*), z kolei minutę dzielimy na *60* części — *60* sekund kątowych (*1″ = 1/60′ = 1/3600°*). Zamiany radianów (`x`) na stopnie dokonujemy według wzoru $\frac{x}{\pi}\cdot 180$. Następnie otrzymany ułamek dziesiętny przedstawimy w postaci sumy części całkowitej *stopni* (`deg`) i dwóch ułamków *min/60* (`min` — całkowita liczba *minut kątowych*, od *0* do *59*), *sek/3600* (`sek` — całkowita liczba *sekund kątowych*, od *0* do *59*). Wynik zapisujemy w postaci ciągu znaków, np. *45°30′15″*, co odpowiada liczbie $45+\frac{30}{60}+\frac{15}{3600}°$ (liczba dziesiętna *45,50416666666667°*).

```
/** Zwraca łańcuch znaków (o postaci 45°30'15") przedstawiający miarę kąta
 * reprezentowanego przez obiekt wyrażoną w stopniach, minutach
 * i sekundach.
 */
   @Override public String toString() {
        if (this.x == 0)
            return "0\u00B0";
        double x = this.x/Math.PI*180;
        StringBuffer str = new StringBuffer();
        if (x < 0) {
            str.append("-");
            x = -x;
        }
        int deg = (int)Math.floor(x);
        x = (x-Math.floor(x))*60;
        int min = (int)Math.floor(x);
        x = (x-Math.floor(x))*60;
        int sek = (int)Math.floor(x+0.5);
        if (sek == 60) {
            sek = 0;
            ++min;
        }
        if (min == 60) {
            min = 0;
            ++deg;
        }
        if (deg != 0) {
            str.append(deg);
            str.append("\u00B0");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        }
        if (min != 0 || (deg != 0 && sek != 0))
            str.append(min).append("\'");
        if (sek != 0)
            str.append(sek).append("\"");
        return str.toString();
    }
```

Zwróćmy uwagę na warunkowe pomijanie zerowych wartości stopni, minut lub sekund w wyjściowym łańcuchu oraz indywidualne potraktowanie przypadku kąta *0 rad = 0°*. Jeśli w wyniku zaokrąglenia otrzymamy *60″*, to zamieniamy tę wartość na *1′* i dodajemy do liczby sekund. Teraz może zdarzyć się, że otrzymamy *60′* — zamienimy to na *1°* i skorygujemy liczbę stopni. Przykładowe obliczenia wykonamy dla kątów $\alpha = \frac{\pi}{4} rad$ i $\beta = \pm 1\, rad$.

```
/** Zadanie Z21.3 */
public class Z21_3 {
    public static void main(String args[]) {
        Angle alfa = new Angle(Math.PI/4);
        System.out.println("alfa = "+alfa.radian()+" rad");
        System.out.println("alfa = "+alfa);
        Angle beta = new Angle(1.0);
        System.out.println("beta = "+beta.radian()+" rad");
        System.out.println("beta = "+beta);
        beta = new Angle(-1.0);
        System.out.println("beta = "+beta.radian()+" rad");
        System.out.println("beta = "+beta);

    }
}
```

Zadanie 21.4. Z21_4.java, Angle.java

Tworzymy trzy kolejne wersje konstruktora `Angle`. Szczegółowe komentarze dokumentacyjne ułatwią zrozumienie kodu konstruktorów. Zasadę miary stopniowej kąta przybliżono w rozwiązaniu zadania 21.3.

```
/** Tworzy nowy obiekt <code>Angle</code> reprezentujący kąt o podanej
 * liczbie (całkowitej) stopni z przedziału &lt;0, 360&gt;.
 * @param deg miara kąta w stopniach, liczba całkowita typu
 * <code>int</code> z przedziału &lt;0, 360&gt;.
 * @throws IllegalArgumentException, gdy liczba stopni nie mieści się
 * w przedziale &lt;0, 360&gt;.
 */
    public Angle(int deg) {
        if (deg < 0 || deg > 360)
            throw new IllegalArgumentException("Liczba stopni poza zakresem.");
        x = deg/180.0*Math.PI;
    }

/** Tworzy nowy obiekt <code>Angle</code> reprezentujący kąt o podanej
 * liczbie stopni i minut.
 * @param deg liczba stopni, liczba całkowita typu <code>int</code>
 * z przedziału &lt;0, 360&gt;,
 * @param min liczba minut, liczba całkowita typu <code>int</code>
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
     * z przedziału &lt;0, 60&gt;.
     * @throws IllegalArgumentException, gdy liczba stopni nie mieści się
     * w przedziale &lt;0, 360&gt;,
     * @throws IllegalArgumentException, gdy liczba minut nie mieści się
     * w przedziale &lt;0, 60&gt;.
     */
        public Angle(int deg, int min) {
            if (deg < 0 || deg > 360)
                throw new IllegalArgumentException("Liczba stopni poza zakresem.");
            if (min < 0 || min > 60)
                throw new IllegalArgumentException("Liczba minut poza zakresem.");
            x = (deg+min/60.0)/180.0*Math.PI;
        }

    /** Tworzy nowy obiekt <code>Angle</code> reprezentujący kąt o podanej
     * liczbie stopni, minut i sekund.
     * @param deg liczba stopni, liczba całkowita typu <code>int</code>
     * z przedziału &lt;0, 360&gt;,
     * @param min liczba minut, liczba całkowita typu <code>int</code>
     * z przedziału &lt;0, 60&gt;,
     * @param sek liczba sekund, liczba całkowita typu <code>int</code>
     * z przedziału &lt;0, 60&gt;.
     * @throws IllegalArgumentException, gdy liczba stopni nie mieści się
     * w przedziale &lt;0, 360&gt;,
     * @throws IllegalArgumentException, gdy liczba minut nie mieści się
     * w przedziale &lt;0, 60&gt;,
     * @throws IllegalArgumentException, gdy liczba sekund nie mieści się
     * w przedziale &lt;0, 60&gt;.
     */
        public Angle(int deg, int min, int sek) {
            if (deg < 0 || deg > 360)
                throw new IllegalArgumentException("Liczba stopni poza
                    zakresem.");
            if (min < 0 || min > 60)
                throw new IllegalArgumentException("Liczba minut poza
                    zakresem.");
            if (sek < 0 || sek > 60)
                throw new IllegalArgumentException("Liczba sekund poza
                    zakresem.");
            x = (deg+min/60.0+sek/3600.0)/180.0*Math.PI;
        }
```

Obliczenia przykładowe podamy dla kilku kątów, np. *15°*, *22°30′* i *10°30″* (zero minut pominęliśmy w tym zapisie, ale musimy przekazać do metody wartość *0* jako drugi parametr).

```
/** Zadanie Z21.4 */
public class Z21_4 {
    public static void main(String args[]) {
        Angle alfa = new Angle(15);
        System.out.print("alfa = "+alfa);
        System.out.println(" = "+alfa.radian()+" rad");

        alfa = new Angle(22, 30);
        System.out.print("alfa = "+alfa);
        System.out.println(" = "+alfa.radian()+" rad");

        alfa = new Angle(10, 0, 30);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.print("alfa = "+alfa);
        System.out.println(" = "+alfa.radian()+" rad");
    }
}
```

Zadanie 21.5. Z21_5.java, Angle.java

Wyznaczamy położenie znaków stopnia (`int p = st.indexOf("\u00B0")`), minuty (`int q = st.indexOf("\'")`) i sekundy kątowej (`int r = st.indexOf("\"")`) w ciągu wejściowym. Otrzymamy indeksy wskazujące położenie tych znaków w łańcuchu `st` lub wartość `-1`, gdy znak nie występuje w łańcuchu. Na tej podstawie możemy wstępnie ocenić poprawność ciągu znaków. Pominiemy tę analizę i spróbujemy wyznaczyć liczbę stopni, minut i sekund, konwertując odpowiedni podciąg na liczbę całkowitą. Czynności te ujmujemy w bloku `try {...}` i przechwytujemy ewentualne wyjątki. Przyczyną zgłoszenia wyjątku może być brak jednego z symboli (°, ' lub "), co spowoduje podanie parametru `-1` do metody `substring()` i wygenerowanie wyjątku `java.lang.StringIndexOutOfBoundsException`. Drugim powodem zgłoszenia wyjątku może być błąd w zapisie liczb, np. *2O°30'15"* (litera *O* zamiast cyfry *0*) — metoda `parseInt()` zgłosi wyjątek `java.lang.NumberFormatException`. Przechwycimy te wyjątki i zastąpimy je wyjątkiem `throw new IllegalArgumentException(st)`.

Jeśli dane wejściowe są poprawne, to możemy zamienić odczytane wartości `deg`, `min` i `sek` na miarę kąta w radianach i podstawić uzyskaną wartość do pola `x` obiektu: `this.x = (deg+min/60.0+sek/3600.0)/180.0*Math.PI;`.

```
public Angle(String st) {
    int deg, min, sek;
    int p = st.indexOf("\u00B0"); // stopień
    int q = st.indexOf("\'");     // minuta kątowa
    int r = st.indexOf("\"");     // sekunda kątowa
    try {
        deg = Integer.parseInt(st.substring(0, p));
        min = Integer.parseInt(st.substring(p+1, q));
        sek = Integer.parseInt(st.substring(q+1, r));
    } catch (Exception e) {
        throw new IllegalArgumentException(st);
    }
    this.x = (deg+min/60.0+sek/3600.0)/180.0*Math.PI;
}
```

Ta wersja konstruktora ma co najmniej trzy wady:

1. Nie dopuszcza skróconych postaci danych, np. *25°15'* lub *33°* (należy napisać *25°15'0"* lub *33°0'0"*).
2. Pozwala na dziwne postacie danych, np. *13°–13'17"* (ujemna liczba minut) lub *1°360'99"* (zbyt duża liczba minut i sekund) — dane te zostaną zinterpretowane jako *12°47'17"* (od *13°* odjęto *13'* i wyszło *12°47'*) lub *7°1'39"* (*360' = 6°*, *99" = 1'39"* — ogólny bilans się zgadza).
3. Źle interpretuje ujemną miarę kąta, np. *22°30'0" = 0.39269908169872414 rad*, ale *–22°30'0"* zostanie zinterpretowane jako *–21°30'0" = –0.3752457891787809 rad*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Spróbujemy te wady wyeliminować. Zainicjowanie zmiennej `sek = 0` i dodanie instrukcji warunkowej:

```
if (r > 0)
    sek = Integer.parseInt(st.substring(q+1, r));
```

pozwoli pomijać w ciągu danych liczbę sekund i symbol sekundy ("). Dopuszczalna jest postać danych bez sekund, np. *22°30'*. Zainicjujmy dodatkowo `min = 0` i dodajmy instrukcje warunkowe:

```
if (q > 0) {
    min = Integer.parseInt(st.substring(p+1, q));
    if (r > 0)
        sek = Integer.parseInt(st.substring(q+1, r));
}
```

Teraz możemy wprowadzać dane w postaci *22°15'37"*, *22°15'* lub *22°*. Nie można pominąć minut, gdy występują sekundy — kąt *15°30"* musimy zapisać jako *15°0'30"* — oraz nie można pominąć stopni — musimy pisać *0°45'* zamiast *45'*. To już nie jest zbyt duża niedogodność.

Liczba minut i liczba sekund powinny przyjmować wartości wyłącznie dodatnie z zakresu od *0* do *59*. Fakt ten sprawdzimy przed przeliczaniem podanej miary kąta na radiany:

```
if ((min < 0) || (min > 59) || (sek < 0) || (sek > 59))
    throw new IllegalArgumentException(st);
```

i zgłosimy wyjątek, gdy dane nie będą poprawne.

Problem ujemnej miary kąta możemy rozwiązać tak:

```
boolean znak =false;
if (deg < 0) {
    znak = true;
    deg = -deg;
}
this.x = (deg+min/60.0+sek/3600.0)/180.0*Math.PI;
if (znak)
    this.x = -this.x;
```

Po tych zmianach kod konstruktora będzie wyglądał następująco:

```
public Angle(String st) {
    int deg, min = 0, sek = 0;
    int p = st.indexOf("\u00B0"); // stopień
    int q = st.indexOf("\'");     // minuta kątowa
    int r = st.indexOf("\"");     // sekunda kątowa
    try {
        deg = Integer.parseInt(st.substring(0, p));
        if (q > 0) {
            min = Integer.parseInt(st.substring(p+1, q));
            if (r > 0)
                sek = Integer.parseInt(st.substring(q+1, r));
        }
    } catch (Exception e) {
        throw new IllegalArgumentException(st);
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        if ((min < 0) || (min > 59) || (sek < 0) || (sek > 59))
            throw new IllegalArgumentException(st);
        boolean znak =false;
        if (deg < 0) {
            znak = true;
            deg = -deg;
        }
        this.x = (deg+min/60.0+sek/3600.0)/180.0*Math.PI;
        if (znak)
            this.x = -this.x;
    }
```

Działanie konstruktora możemy sprawdzić dla kilku przykładowych łańcuchów znaków (poprawnych lub błędnych).

```
/** Zadanie Z21.5 */
public class Z21_5 {
    public static void main(String args[]) {
        String[] angles = { "15°30'27\"", "43°52'", "22°", "-22°30'",
            "22°30'", "35'30\"", "1°65'", "abc", "36", "15°-30'" };
        Angle alfa = null;
        for(String a: angles) {
            try {
                alfa = new Angle(a);
                System.out.print("alfa = "+alfa);
                System.out.println(" = "+alfa.radian()+" rad");
            } catch (IllegalArgumentException e){
                System.out.println("Błędne dane: "+a);
            }
        }
    }
}
```

Zadanie 21.6. Z21_6.java, Angle.java

Funkcje trygonometryczne są funkcjami okresowymi w swojej dziedzinie (zbiorze liczb rzeczywistych). Funkcje odwrotne możemy określić w sposób jednoznaczny wyłącznie w wybranych przedziałach.

Zwykle przyjmujemy takie dziedziny funkcji odwrotnych: $\left\langle -\frac{\pi}{2}, \frac{\pi}{2} \right\rangle$ dla funkcji *arcus sinus* i *arcus cosecans*, $\langle 0, \pi \rangle$ dla *arcus cosinus* i *arcus secans*, $\left(-\frac{\pi}{2}, \frac{\pi}{2} \right)$ dla funkcji *arcus tangens* i $(0, \pi)$ dla *arcus cotangens*.

W obliczeniach wykorzystamy metody z klasy `Math`: `asin()` (*arcus sinus*), `acos()` (*arcus cosinus*) i `atan()` (*arcus tangens*) oraz znane właściwości funkcji trygonometrycznych i funkcji do nich odwrotnych.

Funkcje *sinus* i *cosinus* przyjmują wartości z przedziału $\langle -1, 1 \rangle$, więc argumentem funkcji odwrotnej mogą być wyłącznie liczby z tego przedziału. Funkcjom *arcus sinus* i *arcus cosinus* odpowiadają metody:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public void setOfSin(double x) {
    if (x >= -1 && x <= 1)
        this.x = Math.asin(x);
    else
        throw new IllegalArgumentException();
}

public void setOfCos(double x) {
    if (x >= -1 && x <= 1)
        this.x = Math.acos(x);
    else
        throw new IllegalArgumentException();
}
```

Zbiór wartości funkcji tangens i cotangens jest zbiorem wszystkich liczb rzeczywistych, czyli argumenty funkcji odwrotnych nie podlegają ograniczeniom.

```
public void setOfTan(double x) {
    this.x = Math.atan(x);
}
public void setOfCot(double x) {
    if (x == 0.0)
        this.x = Math.PI/2;
    else
        this.x = Math.PI+Math.atan(1/x);
}
```

Funkcje *secans* i *cosecans* przyjmują wartości ze zbioru $(-\infty, -1\rangle \cup \langle 1, +\infty)$, więc argumenty funkcji odwrotnych nie mogą należeć do przedziału $(-1, 1)$.

```
public void setOfSec(double x) {
    if (x > -1 && x < 1)
        throw new IllegalArgumentException();
    else
        this.x = Math.acos(1/x);
}

public void setOfCsc(double x) {
    if (x > -1 && x < 1)
        throw new IllegalArgumentException();
    else
        this.x = Math.asin(1/x);
}
```

Tworząc program demonstrujący działanie tych metod wybieramy takie wartości argumentu x, dla których wyświetlone wyniki będą łatwe do zweryfikowania.

```
/** Zadanie Z21.6 */
public class Z21_6 {
    public static void main(String args[]) {
        Angle alfa = new Angle(0);;
        double x = Math.sqrt(2)/2;
        alfa.setOfSin(x);
        System.out.printf("sin x = %.6f, ", x);
        System.out.println("alfa = "+alfa);
        alfa.setOfCos(-x);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.printf("cos x = %.6f, ", -x);
        System.out.println("alfa = "+alfa);
        x = -1;
        alfa.setOfTan(x);
        System.out.printf("tan x = %.6f, ", x);
        System.out.println("alfa = "+alfa);
        alfa.setOfCot(x);
        System.out.printf("cot x = %.6f, ", x);
        System.out.println("alfa = "+alfa);
        x = 2/Math.sqrt(3);
        alfa.setOfSec(x);
        System.out.printf("sec x = %.6f, ", x);
        System.out.println("alfa = "+alfa);
        alfa.setOfCsc(x);
        System.out.printf("csc x = %.6f, ", x);
        System.out.println("alfa = "+alfa);
    }
}
```

Weźmy znaną wartość $\sin 60° = \sin\frac{\pi}{3} = \frac{\sqrt{3}}{2} = \cos\frac{\pi}{6} = \cos 30°$. Stąd możemy obliczyć $\csc 60° = \csc\frac{\pi}{3} = \frac{2}{\sqrt{3}} = \sec\frac{\pi}{6} = \sec 30°$. Wykonując powyższy program, uzyskamy taką odpowiedź (dla metod `setOfSec()` i `setOfCsc()`), zgodną z oczekiwaniem:

```
sec x = 1,154701, alfa = 30°
csc x = 1,154701, alfa = 60°
```

Zadanie 21.7. Z21_7.java, Angle.java

W klasie `Math` mamy metodę `atan2()` z dwoma parametrami `x` i `y`, które są współrzędnymi punktu *P* w prostokątnym układzie współrzędnych. Współrzędne tego punktu możemy zapisać w układzie biegunowym w postaci (r, θ), gdzie $x = r \cdot \cos\theta$ i $y = r \cdot \sin\theta$. Do obliczenia miary kąta θ służy właśnie metoda `atan2()`. Zwracany kąt należy do przedziału od $-\pi$ do π (miara kąta ma znak parametru `y`). W metodzie korygujemy ten wynik dla ujemnych wartości parametru `y`, dodając 2π (podwójny okres funkcji *tangens*). Ostateczny wynik należy do przedziału od *0* do 2π (od *0°* do *360°*).

```
public void setOfPoint(double x, double y) {
    if (x == 0 && y == 0)
        throw new IllegalArgumentException();
    if (y >= 0)
        this.x = Math.atan2(y, x);
    else
        this.x = 2*Math.PI+Math.atan2(y, x);
}
```

Przykład działania metody `setOfPoint()`:

```
/** Zadanie Z21.7 */
public class Z21_7 {
    public static void main(String args[]) {
        Angle a = new Angle(0);
        double x = 5.0, y = 5.0;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        a.setOfPoint(x, y);
        System.out.printf("P(%.2f, %.2f), ", x, y);
        System.out.println("kąt "+a.degree()+"\u00B0");
        x = 0.0; y = -5.0;
        System.out.printf("P(%.2f, %.2f), ", x, y);
        a.setOfPoint(x, y);
        System.out.println("kąt "+a.degree()+"\u00B0");
    }
}
```

Zadanie 21.8. Z21_8.java, Angle.java

Kod konstruktora będzie podobny do kodu metody `setOfPoint()` przedstawionej w rozwiązaniu zadania 21.7.

```
public Angle(double x, double y) {
    if (x == 0 && y == 0)
        throw new IllegalArgumentException();
    if (y >= 0)
        this.x = Math.atan2(y, x);
    else
        this.x = 2*Math.PI+Math.atan2(y, x);
}
```

Działanie konstruktora pokażemy na przykładzie dwóch punktów *A(3, 4)* i *B(-5, 0)*. Pomimo całkowitych wartości współrzędnych punktów musimy pamiętać o konieczności ich przedstawienia w postaci liczb zmiennoprzecinkowych (kompilator na podstawie typów parametrów wywołania konstruktora dobierze właściwy kod konstruktora).

```
/** Zadanie Z21.8 */
public class Z21_8 {
    public static void main(String args[]) {
        // punkt A(4.0, 3.0)
        Angle a = new Angle(4.0, 3.0);
        System.out.println("A(4.0, 3.0), kąt "+a);
        // punkt B(-5.0, 0.0)
        Angle b = new Angle(-5.0, 0.0);
        System.out.println("B(-5.0, 0.0), kąt "+b);

    }
}
```


Uwaga

W aktualnej postaci klasy `Angle` dla punktu `A(4, 3)` poprawne będą konstrukcje `Angle a = new Angle(4.0, 3.0)`, `Angle a = new Angle(4.0, 3)` i `Angle a = new Angle(4, 3.0)`, którym odpowiada kąt *36°52'12"*. Natomiast konstrukcja `Angle a = new Angle(4, 3)` zbuduje kąt *4°3'*, korzystając z konstruktora `Angle(int, int)` pochodzącego z rozwiązania zadania 21.4.

Zadanie 21.9. Z21_9.java, Angle.java

We wszystkich utworzonych metodach statycznych `valueOf()` na podstawie parametrów wywołania utworzymy obiekt klasy `Angle` i zwrócimy referencję do tego obiektu. W zależności od typu i liczby parametrów wejściowych wykorzystamy odpowiedni konstruktor.

Ebookpoint.pl kopia dla: pawher@o2.pl

Metoda zwraca obiekt `Angle` reprezentujący kąt `x` radianów.

```
public static Angle valueOf(double x) {
    return new Angle(x);
}
```

Metoda zwraca obiekt reprezentujący kąt o podanej całkowitej liczbie stopni. Możemy zamienić podaną liczbę stopni `deg` na radiany i wywołać podstawowy konstruktor `Angle(double)`, znany z zadania 20.1 (nie ma kontroli zakresu dla zmiennej `deg`):

```
public static Angle valueOf(int deg) {
    return new Angle(deg/180.0*Math.PI);
}
```

lub wykorzystać konstruktor z zadania 21.4 (użyty konstruktor sprawdzi, czy zmienna `deg` mieści się w zakresie od *0°* do *360°*):

```
public static Angle valueOf(int deg) {
    return new Angle(deg);
}
```

Kolejne metody statyczne zbudujemy według tego samego schematu:

```
public static Angle valueOf(int deg, int min) {
    return new Angle((deg+min/60.0)/180.0*Math.PI);
}
public static Angle valueOf(int deg, int min, int sek) {
    return new Angle((deg+min/60.0+sek/3600.0)/180.0*Math.PI);
}
```

lub:

```
public static Angle valueOf(int deg, int min) {
    return new Angle(deg, min);
}
public static Angle valueOf(int deg, int min, int sek) {
    return new Angle(deg, min, sek);
}
```

Zamianę ciągu znaków (postaci *xxx°yy'zz″*) na obiekt klasy `Angle` możemy zrealizować, używając metody statycznej wykorzystującej odpowiedni konstruktor:

```
public static Angle valueOf(String st) {
    return new Angle(st);
}
```

Ponieważ ten konstruktor ma pewne ograniczenia (omówione w rozwiązaniu zadania 21.5), możemy spróbować rozwiązać ten problem inaczej. Szczegółową analizę kodu pozostawiamy Czytelnikowi.

```
public static Angle valueOf(String st) {
    int deg = 0, min = 0, sek = 0;
    int pst = st.indexOf("\u00B0");       // stopień
    if (pst == -1) {                      // nie ma symbolu stopnia
        int pmin = st.indexOf("\'");      // minuta kątowa
        if (pmin == -1) {                 // nie ma symbolu minuty
            int psek = st.indexOf("\""); // sekunda kątowa
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            if (psek == -1) {              // nie ma symbolu sekundy
                throw new IllegalArgumentException();
            } else                         // tylko sekundy
               sek = Integer.parseInt(st.substring(pmin+1, psek));
        } else {
            min = Integer.parseInt(st.substring(0, pmin));
            int psek = st.indexOf("\""); // sekunda kątowa
            if (psek != -1)
               sek = Integer.parseInt(st.substring(pmin+1, psek));
        }
    } else { // jest symbol stopnia
        deg = Integer.parseInt(st.substring(0, pst));
        int pmin = st.indexOf("\'");     // minuta kątowa
        if (pmin == -1) {
            int psek = st.indexOf("\""); // sekunda kątowa
            if (psek != -1)
                sek = Integer.parseInt(st.substring(pst+1, psek));
            } else {
                min = Integer.parseInt(st.substring(pst+1, pmin));
                int psek = st.indexOf("\""); // sekunda kątowa
                if (psek != -1)
                    sek = Integer.parseInt(st.substring(pmin+1, psek));
            }
    }
    return new Angle(deg, min, sek);
}
```

W tej postaci metoda akceptuje następujące ciągi znaków: *15°*, *15°33′*, *15°33′25″*, *15°25″*, *33′*, *33′25″*, *25″*. Za błędne uznaje ciągi cyfr bez jakichkolwiek symboli (tylko liczba bez znaku stopnia, minuty lub sekundy), niebędące liczbami (zawierające litery lub inne znaki niedozwolone w zapisie liczb), oraz poprawnie zbudowane ciągi znaków ze znakiem minus na początku, np. *–22°30′*. Ten ostatni przypadek wynika z ograniczeń narzuconych przez użyty konstruktor (liczba stopni od *0* do *360*) — kostrutor ten jednak sprawdza poprawność liczby minut i sekund, więc nie będziemy z niego rezygnować.

Wprowadzimy poprawki rozszerzające zakres liczby kątów na wartości ujemne od *–360* do *0*. Instrukcję `return new Angle(deg, min, sek)` zastąpimy następującym ciągiem instrukcji:

```
boolean minus = false;
if (deg < 0) {
    deg = -deg;
    minus = true;
}
Angle tmp = new Angle(deg, min, sek);
if (minus)
    tmp.x = -tmp.x;
return tmp;
```

W przypadku błędnych danych metoda rzuca wyjątek `IllegalArgumentException`, gdy wejściowy łańcuch znaków nie odpowiada poprawnej mierze kąta, lub `NumberFormatException`, gdy w wejściowym łańcuchu (pomiędzy symbolami jednostek) znajduje się niepoprawna postać liczby całkowitej.

Ebookpoint.pl kopia dla: pawher@o2.pl

Pozostaje jeszcze metoda zwracająca obiekt — kąt wyznaczony przez punkt *P(x, y)*, środek układu współrzędnych i oś *OX*:

```
public static Angle valueOf(double x, double y) {
    return new Angle(x, y);
}
```

Działanie zdefiniowanych metod sprawdzimy, pisząc program:

```
/** Zadanie Z21.9 */
public class Z21_9 {
    public static void main(String args[]) {
        Angle a;
        a = Angle.valueOf(Math.PI/4);
        System.out.println("a = "+a);
        a = Angle.valueOf(25);
        System.out.println("a = "+a);
        a = Angle.valueOf(25, 30);
        System.out.println("a = "+a);
        a = Angle.valueOf(25, 30, 17);
        System.out.println("a = "+a);

        String[] angles = { "33°", "33°52'", "33°52'17\"", "33°30\"",
            "52'17\"", "17\"", "1°65'", "2o°13'", "36", "15°-30'" };
        Angle alfa = null;
        for(String x: angles) {
            try {
                alfa = Angle.valueOf(x);
                System.out.print("alfa = "+alfa);
                System.out.println(" = "+alfa.radian()+" rad");
            } catch (Exception e){
                System.out.println("Błędne dane: "+x+" "+e);
            }
        }
        a = Angle.valueOf(0.5, Math.sqrt(3)/2);
        System.out.println("a = "+a);
    }
}
```

Zadanie 21.10. Z21_10.java, Angle.java

Dodawanie lub odejmowanie obiektów sprowadza się do wykonania odpowiedniego działania na polu `x` tych obiektów i zwrócenia wyniku.

```
public Angle add(Angle x) {
    return new Angle(this.x+x.x);
}
public Angle sub(Angle x) {
    return new Angle(this.x-x.x);
}
```

Przykład ilustrujący działanie metod:

```
/** Zadanie Z21.10 */
public class Z21_10 {
    private static void writeln(String s, Angle x) {
        StringBuilder w = new StringBuilder(s);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        w.append(" = ").append(x.radian()).append(" rad");
        w.append(" = ").append(x.degree()).append("\u00B0");
        System.out.println(w);
    }
    public static void main(String args[]) {
        Angle a = new Angle(Math.PI/3);
        Angle b = new Angle(Math.PI/6);
        writeln("a", a);
        writeln("b", b);
        writeln("a+b", a.add(b));
        writeln("a-b", a.sub(b));
    }
}
```


Uwaga

Zbudowaną tu metodę `writeln()` wykorzystamy w kilku kolejnych zadaniach (kopiując kod metody do kodu rozwiązania zadania).

Zadanie 21.11. Z21_11.java, Angle.java

W metodach statycznych `sum()` i `diff()` służących do dodawania i odejmowania obiektów `Angle` (miar kątów) wykorzystamy jeden z tych obiektów, metody `add()` i `sub()` (pokazane w rozwiązaniu zadania 21.10) i drugi obiekt podany jako parametr.

```
public static Angle sum(Angle x, Angle y) {
    return x.add(y);
}
public static Angle diff(Angle x, Angle y) {
    return x.sub(y);
}
```

Przykład ilustrujący działanie metod statycznych:

```
/** Zadanie Z21.11 */
public class Z21_11 {
    /* Tu wstaw kod metody: writeln */
    public static void main(String args[]) {
        Angle a = new Angle(Math.PI/2);
        Angle b = new Angle(Math.PI/6);
        writeln("a", a);
        writeln("b", b);
        writeln("a+b", Angle.sum(a, b));
        writeln("a-b", Angle.diff(a, b));
    }
}
```

Kod metody `writeln` pokazano w rozwiązaniu zadania 21.11.

Zadanie 21.12. Z21_12.java, Angle.java

Mnożenie miary kąta (obiektu) przez liczbę zmiennoprzecinkową lub całkowitą (wielokrotność kąta):

```
public Angle mult(double a) {
    return new Angle(a*this.x);
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public Angle mult(int n) {
    return new Angle(n*this.x);
}
```

Dzielenie miary kąta (obiektu) przez liczbę:

```
public Angle div(double a) {
    if (a == 0.0)
        throw new ArithmeticException("Dzielenie przez 0!");
    return new Angle(a*this.x/a);
}
public Angle div(int n) {
    if (n == 0)
        throw new ArithmeticException("Dzielenie przez 0!");
    return new Angle(this.x/n);
}
```

Przykład działania metod `mult()` i `div()`:

```
/** Zadanie Z21.12 */
public class Z21_12 {
    /* Tu wstaw kod metody: writeln */
    public static void main(String args[]) {
        Angle a = new Angle("22°30'");
        writeln("a", a);
        writeln("2a", a.mult(2));
        writeln("4a", a.mult(4));
        writeln("1.5a", a.mult(1.5));
        writeln("a/2", a.div(2));
    }
}
```

Wyniki będą wyświetlanie w postaci *1.5a = 0.5890486225480862 rad = 33.75°*. Jeśli chcemy, aby miara stopniowa była podawana z minutami i sekundami, to należy w kodzie metody `writeln()` zamienić wiersz:

```
w.append(" = ").append(x.degree()).append("\u00B0");
```

na następujący:

```
w.append(" = ").append(x.toString());
```

Zadanie 21.13. Z21_13.java, Angle.java

W metodach statycznych `prod()` i `quot()` służących do dzielenia obiektów `Angle` (miar kątów) przez liczbę wywołamy metody `mult()` i `div()` (pokazane w rozwiązaniu zadania 21.12) dla obiektu podanego jako pierwszy parametr. Drugi parametr (liczbę) przekażemy jako parametr do metod `mult()` i `div()`.

```
public static Angle prod(Angle x, double y) {
    return x.mult(y);
}
public static Angle prod(Angle x, int n) {
    return x.mult(n);
}
public static Angle quot(Angle x, double y) {
    return x.div(y);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
}
public static Angle quot(Angle x, int n) {
    return x.div(n);
}
```

Przykład działania metod statycznych `prod()` i `quot()`:

```
/** Zadanie Z21.13 */
public class Z21_13 {
    /* Tu wstaw kod metody: writeln */
    public static void main(String args[]) {

        Angle a = new Angle("11°15'");
        writeln("a", a);
        writeln("2a", Angle.prod(a, 2));
        writeln("4a", Angle.prod(a, 4));
        writeln("1.5a", Angle.prod(a, 1.5));
        writeln("a/3", Angle.quot(a, 2));
    }
}
```

Zadanie 21.14. Z21_14.java, Angle.java

Definiujemy publiczne i statyczne stałe typu obiektowego i nadajemy im odpowiednie wartości. Ze względu na użycie słowa `final` nie będziemy mogli tym stałym przypisać referencji do innych obiektów, np. `Angle RADIAN = new Angle(2.0)`. Niestety, możliwość zmiany pola obiektu przy użyciu metod `setOfXxx` (zob. zadania 21.6 i 21.7) jest nadal aktualna.

```
public static final Angle RIGHT_ANGLE = new Angle(Math.PI/2);
public static final Angle STRAIGHT_ANGLE = new Angle(Math.PI);
public static final Angle FULL_ANGLE = new Angle(2*Math.PI);
public static final Angle RADIAN = new Angle(1.0);
public static final Angle DEGREE = new Angle(1);
public static final Angle ARCMINUTE = new Angle(0, 1);
public static final Angle ARCSECOND = new Angle(0, 0, 1);
```

W przykładzie wyświetlimy wartości miar kątów reprezentowanych przez zdefiniowane stałe.

```
/** Zadanie Z21.14 */
public class Z21_14 {
    public static void main(String args[]) {
        System.out.println("Kąt prosty: "+Angle.RIGHT_ANGLE);
        System.out.println("Kąt półpełny: "+Angle.STRAIGHT_ANGLE);
        System.out.println("Kąt pełny: "+Angle.FULL_ANGLE);
        System.out.println("1 radian: "+Angle.RADIAN);
        System.out.println("PI radian: "+Angle.RADIAN.mult(Math.PI));
        System.out.println("1 stopień: "+Angle.DEGREE);
        System.out.println("1 minuta kątowa: "+Angle.ARCMINUTE);
        System.out.println("1 sekunda kątowa: "+Angle.ARCSECOND);
    }
}
```

Zwróćmy uwagę na możliwość wykonywania obliczeń (przy użyciu metod) związanych z tymi stałymi, np. `Angle.RADIAN.mult(Math.PI)`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 21.15. Z21_15.java, Angle.java

Kąty dopełniające mają wspólne jedno ramię i ich suma jest kątem prostym (co możemy wyrazić wzorem $\alpha + \beta = 90°$, czyli $\beta = 90° - \alpha$).

```
public static Angle compl(Angle x) {
    if (x.x < 0.0 && x.x > RIGHT_ANGLE.x)
        throw new IllegalArgumentException();
    return RIGHT_ANGLE.sub(x);
}
```

Kąty przyległe mają wspólne jedno ramię i ich suma jest kątem półpełnym ($\alpha + \beta = 180°$, czyli $\beta = 180° - \alpha$).

```
public static Angle suppl(Angle x) {
    if (x.x < 0.0 && x.x > STRAIGHT_ANGLE.x)
        throw new IllegalArgumentException();
    return STRAIGHT_ANGLE.sub(x);
}
```

Przykład obliczeń z zastosowaniem metod `compl()` i `suppl()`:

```
/** Zadanie Z21.15 */
public class Z21_15 {
    public static void main(String args[]) {
        Angle alfa = new Angle(27, 32, 15);
        System.out.println("alfa = "+alfa);
        System.out.println("Kąt dopełniający do alfa: "+
            Angle.compl(alfa));
        System.out.println("Kąt przyległy do alfa: "+Angle.suppl(alfa));
    }
}
```

Zadanie 21.16. Z21_16.java

Do wczytywania danych wykorzystamy obiekt `input` klasy `Scanner`. Najpierw sprawdzimy, czy w strumieniu danych jest liczba całkowita. Jeśli tak, to potraktujemy ją jako miarę kąta wyrażoną w stopniach. W przeciwnym razie sprawdzamy, czy nie jest to liczba zmiennoprzecinkowa. Jeśli wprowadzono liczbę zmiennoprzecinkową, to zamieniamy tę liczbę (traktowaną jako miarę kąta w stopniach) na radiany i wywołujemy odpowiednią metodę. Natomiast gdy w strumieniu nie ma liczby, to spodziewamy się tekstu o postaci *23°35'25"*. Błędny ciąg znaków wygeneruje wyjątek zgłoszony przez konstruktor `Angle(String)`.

```
private static Angle inputAngle(String s) {
    System.out.print("Podaj "+s);
    Scanner input = new Scanner(System.in);
    if (input.hasNextInt())
        return new Angle(input.nextInt());
    else if (input.hasNextDouble())
        return new Angle(Math.toRadians(input.nextDouble()));
    else {
        String str = input.next();
        return new Angle(str);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

W trójkącie równoramiennym kąty przy podstawie (β) mają równe miary. Suma kątów wewnętrznych trójkąta jest kątem półpełnym: $\alpha + 2\beta = 180°$, czyli $2\beta = 180° - \alpha$ (do obliczeń wykorzystamy metodę `suppl()`). Stąd $\beta = \frac{180° - \alpha}{2}$, co możemy wyrazić formułą `Angle beta = Angle.suppl(alfa).div(2)`. Sprawdzenie obliczeń wykonamy, dodając miary wszystkich kątów trójkąta: `beta.mult(2).add(alfa)`.

```
import java.util.Scanner;
/** Zadanie Z21.16 */
public class Z21_16 {
    /* Tu wstaw kod metody inputAngle. */
    public static void main(String args[]) {
        System.out.println("Obliczanie miar kątów w trójkącie
            równoramiennym.");
        Angle alfa = inputAngle("Kąt przy wierzchołku trójkąta
            równoramiennego,\nalfa = ");
        Angle beta = Angle.suppl(alfa).div(2);
        System.out.println("Kąt przy wierzchołku trójkąta
            równoramiennego, alfa = "+alfa);
        System.out.println("Kąt przy podstawie trójkąta równoramiennego,
            beta = "+beta);
        System.out.println("Sprawdzenie - suma kątów w trójkącie:"+
            beta.mult(2).add(alfa));
    }
}
```

Zadanie 21.17. Z21_17.java

Do wprowadzania danych wykorzystamy metodę `inputAngle()` omówioną w rozwiązaniu zadania 21.16. Kąt leżący naprzeciw podstawy w trójkącie równoramiennym obliczymy ze wzoru $\alpha = 180° - 2\beta$, gdzie β jest miarą kąta przyległego do podstawy. Obliczenie to zrealizujemy w następujący sposób: `Angle alfa = Angle.suppl(beta.mult(2))`.

```
import java.util.Scanner;
/** Zadanie Z21.17 */
public class Z21_17 {
    /* Tu wstaw kod metody inputAngle. */
    public static void main(String args[]) {
        System.out.println("Obliczanie miar kątów w trójkącie
            równoramiennym.");
        Angle beta = inputAngle("Kąt przy podstawie trójkąta
            równoramiennego,\nalfa = ");
        Angle alfa = Angle.suppl(beta.mult(2));
        System.out.println("Kąt przy podstawie trójkąta równoramiennego,
            beta = "+beta);
        System.out.println("Kąt przy wierzchołku trójkąta
            równoramiennego, alfa = "+alfa);
        System.out.println("Sprawdzenie - suma kątów w trójkącie:
            "+beta.mult(2).add(alfa));
    }
}
```

Obliczenia sprawdzimy, dodając miary kątów trójkąta: `beta.mult(2).add(alfa)`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 21.18. Z21_18.java

Do wprowadzania danych (długości odcinków) wykorzystamy metodę `inputSide()` (ang. *side* — bok wielokąta).

```
private static Double inputSide(String s) {
    System.out.print("Podaj "+s);
    Scanner input = new Scanner(System.in);
    return input.nextDouble();
}
```

Do wyznaczenia miary kąta możemy zastosować *twierdzenie cosinusów* (będące uogólnieniem *twierdzenia Pitagorasa*), wyrażone wzorem $c^2 = a^2 + b^2 - 2ab\cos\gamma$ (kąt γ leży naprzeciw boku *c*). Stąd obliczymy $\cos\gamma = \frac{a^2 + b^2 - c^2}{2ab}$. Przestawiając cyklicznie długości boków, otrzymamy kolejne wzory: $\cos\alpha = \frac{b^2 + c^2 - a^2}{2bc}$ i $\cos\beta = \frac{c^2 + a^2 - b^2}{2ca}$ (po uporządkowaniu $\cos\beta = \frac{a^2 + c^2 - b^2}{2ac}$).

Podstawiając $b = c$ we wzorze $\cos\alpha = \frac{b^2 + c^2 - a^2}{2bc}$, otrzymamy

$$\cos\alpha = \frac{b^2 + b^2 - a^2}{2bb} = \frac{2b^2 - a^2}{2b^2} = 1 - \frac{a^2}{2b^2}$$

Na podstawie tego wzoru obliczymy:

```
alfa.setOfCos(1-a*a/(2*b*b));
```

Kąt β możemy obliczyć w sposób poznany w rozwiązaniu zadania 21.16:

```
Angle beta = Angle.suppl(alfa).div(2)
```

lub ponownie stosując wzór wynikający z twierdzenia cosinusów:

$$\cos\beta = \frac{c^2 + a^2 - b^2}{2ca} = \frac{b^2 + a^2 - b^2}{2ba} = \frac{a^2}{2ab} = \frac{a}{2b}$$

```
beta.setOfCos(a/2/b);
```

Kod programu przedstawia się następująco:

```
import java.util.Scanner;
/** Zadanie Z21.18 */
public class Z21_18 {
    /* Tu wstaw kod metody inputSide. */
    public static void main(String args[]) {
        System.out.println("Obliczanie miar kątów w trójkącie
            równoramiennym.");
        double a = inputSide("Długość podstawy trójkąta
            równoramiennego,\na = ");
        double b = inputSide("Długość ramienia trójkąta
            równoramiennego,\nb = ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        Angle alfa = new Angle(0);
        alfa.setOfCos(1-a*a/(2*b*b));
        Angle beta = new Angle(0);
        beta.setOfCos(a/2/b);
        System.out.println("Kąt przy wierzchołku trójkąta
            równoramiennego, alfa = "+alfa);
        System.out.println("Kąt przy podstawie trójkąta równoramiennego,
            beta = "+beta);
        System.out.println("Sprawdzenie - suma kątów w trójkącie: "+
            beta.mult(2).add(alfa));
    }
}
```

Zadanie 21.19. Z21_19.java

Do wprowadzania danych (długości odcinków) wykorzystamy metodę `inputSide()` przedstawioną w rozwiązaniu zadania 21.18.

Wysokość opuszczona na podstawę dzieli trójkąt równoramienny na dwa przystające trójkąty prostokątne o przyprostokątnych o długości $\frac{a}{2}$ i h (kąty ostre leżące naprzeciw tych boków mają miary $\frac{\alpha}{2}$ i β). Ponieważ $\tan\frac{\alpha}{2}=\frac{\frac{a}{2}}{h}=\frac{a}{2h}$, to możemy obliczyć miarę kąta α:

```
alfa.setOfTan(a/2/h); // połowa kąta alfa
alfa = alfa.mult(2);  // kąt alfa
```

Podobnie obliczymy $\tan\beta=\frac{h}{\frac{a}{2}}=\frac{2h}{a}$, więc:

```
beta.setOfTan(2*h/a); // kąt beta
```

Oto kompletny kod programu:

```
import java.util.Scanner;
/** Zadanie Z21.19 */
public class Z21_19 {
    /* Tu wstaw kod metody inputSide. */
    public static void main(String args[]) {
        System.out.println("Obliczanie miar kątów w trójkącie równoramiennym.");
        double a = inputSide("Długość podstawy trójkąta równoramiennego,\na = ");
        double h = inputSide("Wysokość trójkąta równoramiennego,\nh = ");
        Angle beta = new Angle(0);
        beta.setOfTan(2*h/a);
        Angle alfa = new Angle(0);
        alfa.setOfTan(a/2/h);
        alfa = alfa.mult(2);
        System.out.println("Kąt przy wierzchołku trójkąta równoramiennego,
            alfa = "+alfa);
        System.out.println("Kąt przy podstawie trójkąta równoramiennego,
            beta = "+beta);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Sprawdzenie - suma kątów w trójkącie: "+
            beta.mult(2).add(alfa));
    }
}
```

Zadanie 21.20. Z21_20.java

Do wprowadzania danych (długości boków trójkąta) zastosujemy metodę `inputSide()` przedstawioną w rozwiązaniu zadania 21.18. Miary kątów obliczymy na podstawie *twierdzenia cosinusów* — wzory omówiono w rozwiązaniu zadania 21.18.

```
import java.util.Scanner;
/** Zadanie Z21.20 */
public class Z21_20 {
    /* Tu wstaw kod metody inputSide. */
    public static void main(String args[]) {
        System.out.println("Obliczanie miar kątów w trójkącie o bokach
            a, b i c.");
        double a = inputSide("Długość boku trójkąta,\na = ");
        double b = inputSide("Długość boku trójkąta,\nb = ");
        double c = inputSide("Długość boku trójkąta,\nc = ");
        Angle alfa = new Angle(0);
        alfa.setOfCos((b*b+c*c-a*a)/(2*b*c));
        Angle beta = new Angle(0);
        beta.setOfCos((a*a+c*c-b*b)/(2*a*c));
        Angle gamma = new Angle(0);
        gamma.setOfCos((a*a+b*b-c*c)/(2*a*b));
        System.out.println("alfa = "+alfa);
        System.out.println("beta = "+beta);
        System.out.println("gamma = "+gamma);
        System.out.println("Sprawdzenie - suma kątów w trójkącie: "+
            alfa.add(beta).add(gamma));
    }
}
```

Po obliczeniu miar dwóch kątów (α i β) miarę trzeciego kąta możemy wyznaczyć ze wzoru $\gamma = 180° - (\alpha + \beta)$:

```
Angle gamma = Angle.diff(Angle.STRAIGHT_ANGLE, Angle.sum(alfa, beta));
```

lub:

```
Angle gamma = Angle.STRAIGHT_ANGLE.sub(alfa.add(beta));
```

22. Liczby rzymskie i klasa Roman

Zadanie 22.1. Z22_1.java

Analizując zapis liczb rzymskich (z zakresu od *1* do *3999*), możemy zauważyć, że symbole *M* (*1000*), *C* (*100*), *X* (*10*) i *I* (*1*) mogą powtarzać się do trzech razy obok siebie, a symbole *D* (*500*), *L* (*50*) i *V* (*5*) występują maksymalnie jeden raz. Ponadto ważne są kombinacje dwóch znaków: *CM* (*900*), *CD* (*400*), *XC* (*90*), *XL* (*40*), *IX* (*9*) i *IV* (*4*).

Ebookpoint.pl kopia dla: pawher@o2.pl

Kod zamieniający liczbę `n` na ciąg znaków w systemie rzymskim jest obszerny, ale niezbyt trudny do zrozumienia. Liczbę `n` zmniejszamy kolejno, jeśli jest to możliwe, o wartości *1000*, *900*, *500*, *400*, *100*, *90*, *50*, *40*, *10*, *9*, *5*, *4*, *1*. W zamian za to do wyniku dodajemy odpowiednio znaki *M*, *CM*, *D*, *CD*, *C*, *XC*, *L*, *XL*, *X*, *IX*, *V*, *IV*, *I*. Znaki *M*, *C*, *X* i *I* mogą wystąpić co najwyżej trzykrotnie (stąd użycie pętli `while`), pozostałe tylko raz. Należy zauważyć, że wystąpienie *CM* wyklucza pojawienie się *D*, *CD* i *C* w dalszej części ciągu znaków, podobnie *D* wyklucza *CD* i *C* oraz *CD* wyklucza *C* (stosujemy zagnieżdżone instrukcje warunkowe). Podobne reguły obowiązują dla zestawu znaków *XC*, *L*, *XL* i *X* oraz *IX*, *V*, *IV* i *I*.

```
StringBuilder tmp = new StringBuilder("");
/* Znaki rzymskie M, CM, D i CD - liczby 1000, 900, 500, 400 */
while (n >= 1000) {
    n -= 1000;
    tmp.append("M");
}
if (n >= 900) {
    n -= 900;
    tmp.append("CM");
} else if (n >= 500) {
    n -= 500;
    tmp.append("D");
} else if (n >= 400) {
    n -= 400;
    tmp.append("CD");
}

/* Znaki rzymskie C, XC, L, XL - liczby 100, 90, 50, 40 */
while (n >= 100) {
    n -= 100;
    tmp.append("C");
}
if (n >= 90) {
    n -= 90;
    tmp.append("XC");
} else if (n >= 50) {
    n -= 50;
    tmp.append("L");
} else if (n >= 40) {
    n -= 40;
    tmp.append("XL");
}

/* Znaki rzymskie X, IX, V, IV - liczby 10, 9, 5, 4 */
while (n >= 10) {
    n -= 10;
    tmp.append("X");
}
if (n >= 9) {
    n -= 9;
    tmp.append("IX");
} else if (n >= 5) {
    n -= 5;
    tmp.append("V");
} else if (n >= 4) {
    n -= 4;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        tmp.append("IV");
    }

    /* Znak I - liczba 1 */
    while (n >= 1) {
        n -= 1;
        tmp.append("I");
    }
```

Obiekt `tmp` zawiera początkową wartość liczby `n` w zapisie rzymskim (aktualna wartość `n` jest zerem).

Pobieraną z klawiatury wartość liczby będziemy przechowywali w zmiennej `liczba` (typu `int`). W programie utworzymy pętlę ze sprawdzaniem warunku na końcu — działanie pętli przerwiemy, gdy zmienna `liczba` będzie miała wartość `0`. W pętli wczytujemy liczbę całkowitą i sprawdzamy, czy jej wartość mieści się w zakresie od `1` do `3999`. Jeśli liczba jest poza tym zakresem, wyświetlamy stosowny komunikat (`"Liczba poza zakresem."`) i przechodzimy (instrukcją `continue`) do sprawdzania warunku powtarzania pętli, pomijając kod zamiany `n` na liczbę w systemie rzymskim.

```
import java.util.Scanner;
/** Zadanie Z22.1 */
public class Z22_1 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Rzymski sposób zapisu liczb od 1 do 3999");
        int liczba;
        do {
            System.out.print("Podaj liczbę (0 - koniec obliczeń), n = ");
            liczba = input.nextInt();
            if (liczba < 1 || liczba > 3999) {
                System.out.println("Liczba poza zakresem.");
                continue;
            }
            int n = liczba;

            /* Tu wstaw kod zamiany n na liczbę w systemie rzymskim */

            System.out.println(liczba+" = "+tmp.toString());
        } while (liczba !=0);
    }
}
```

Stosując dwie tablice z wartościami wybranych liczb rzymskich i odpowiadających im wartości dziesiętnych:

```
private static final String[] rz = {"M", "CM", "D", "CD", "C", "XC", "L", "XL",
"X", "IX", "V", "IV", "I"};
private static final int[] ar = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5,
4, 1};
```

możemy uprościć kodowanie liczb w systemie rzymskim.

```
StringBuilder tmp = new StringBuilder("");
for(int i = 0; i<13; ++i) {
    while (n >= ar[i]) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            n -= ar[i];
            tmp.append(rz[i]);
        }
    }
```

Na początku wartość zmiennej `n` jest równa konwertowanej liczbie i mieści się w zakresie od `1` do `3999`. Elementy w obu tablicach są uporządkowane od wartości największej (*1000*) do najmniejszej (*1*). Bierzemy kolejne wartości i dopóki liczba jest większa od tej wartości, dopóty do ciągu znaków rzymskich dodajemy odpowiedni symbol, a liczbę zmniejszamy o tę wartość. Po dojściu do końca tablicy mamy poprawnie zakodowaną liczbę w systemie rzymskim.

Zadanie 22.2. Z22_2.java

Oprócz tablic (wprowadzonych w rozwiązaniu zadania 22.1):

```
private static final String[] rz = {"M", "CM", "D", "CD", "C", "XC", "L", "XL",
"X", "IX", "V", "IV", "I"};
private static final int[] ar = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5,
4, 1};
```

dodamy tablicę `lp` zawierającą maksymalną liczbę powtórzeń symboli z tablicy `rz`:

```
private static final int[] lp = {3, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 3};
```

oraz tablicę `err` z niedozwolonymi kombinacjami symboli:

```
private static final String[] err = {"CMD", "CMC", "XCL", "XCX", "IXV", "IXI",
"DCD", "CDC", "LXL", "XLX", "VIV", "IVI"};
```

Ciąg symboli `CMD` oznaczałby liczbę 900+500 = 1400, którą poprawnie zapisujemy w postaci `MCD` (1000+400). Kolejny ciąg `CMC` można interpretować jako 900+100 = 1000, ale to nie ma sensu, bo 1000 zapisujemy jednym znakiem `M`. Podobnie jest z pozostałymi ciągami uznanymi za błędne i zapisanymi w tablicy `err`.

Najpierw sprawdzimy, czy podany ciąg znaków (`roman`) nie zawiera niedozwolonych ciągów zapisanych w tablicy `err`. Przeglądając tablicę błędnych sekwencji znaków, sprawdzamy, czy występują w łańcuchu `roman`. Jeśli tak się zdarzy, to metoda `indexOf()` zwróci wartość `-1` i zostanie zgłoszony wyjątek.

```
for (String x: err)
    if (roman.indexOf(x) != -1)
        throw new ArithmeticException("Niewłaściwy ciąg znaków: "+x+
            " w "+roman);
```

Jeśli ciąg wejściowy nie zawierał niedozwolonych sekwencji, możemy ustalić początkową wartość wyniku (`tmp`) i zainicjować indeks analizowanego znaku rzymskiego (pobieranego z tablicy `rz`).

```
int tmp = 0;
int index = 0;  // indeks analizowanego znaku rzymskiego
```

Przeglądamy w pętli (`for (int i = 0; i < 13; i++)`) tablicę `rz`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zerujemy (`int powt = 0`) licznik powtórzeń symbolu `rz[i]`. W pętli (`while`) sprawdzamy, ile razy symbol rzymski (`rz[i]`) pojawia się w ciągu znaków `roman`. Każda pozytywna odpowiedź powoduje zwiększenie zmiennej `tmp` (`tmp += ar[i]`) o wartość (`ar[i]`) odpowiadającą sprawdzanemu symbolowi (`rz[i]`), zwiększenie indeksu o rozmiar symbolu (`index += rz[i].length()`) i zwiększenie licznika powtórzeń (`++powt`). Badany na początku pętli `while` warunek `roman.startsWith(rz[i], index)` sprawdza, czy w łańcuchu `roman` na pozycji o indeksie `index` rozpoczyna się ciąg znaków `rz[i]` — wynik jest wartością logiczną (`true` lub `false`). Po wyjściu z pętli `while` sprawdzamy, czy symbol `rz[i]` nie wystąpił zbyt wiele razy (`if (powt > lp[i])...`). Jeśli liczba powtórzeń przekracza dozwoloną liczbę (zapisaną w tabeli `lp`), to zgłaszany jest wyjątek.

Po wyjściu z pętli `for` dokonujemy jeszcze jednego sprawdzenia:

```
if (index != roman.length())
     throw new ArithmeticException("Niewłaściwa liczba rzymska: "+roman);
```

Rozbieżność pomiędzy wartością zmiennej `index` i długością łańcucha `roman` świadczy o niepoprawnej budowie liczby rzymskiej.

Poniższy kod zamienia łańcuch znaków `roman` na liczbę `tmp` lub zgłasza wyjątek:

```
for (String x: err)
    if (roman.indexOf(x) != -1)
        throw new ArithmeticException("Niewłaściwy ciąg znaków: "+
            x+" w "+roman);
int tmp = 0;     // wartość początkowa wyniku
int index = 0;   // indeks analizowanego znaku
for (int i = 0; i < 13; i++) {
    int powt = 0;
    while (roman.startsWith(rz[i], index)) {
        tmp += ar[i];
        index += rz[i].length();
        ++powt;
    }
if (powt > lp[i])
    throw new ArithmeticException("Za dużo znaków: "+rz[i]+" w "+roman);
}
if (index != roman.length())
    throw new ArithmeticException("Niewłaściwa liczba rzymska: "+roman);
```

W głównej pętli programu (`do {...} while(true);`) wczytujemy liczbę rzymską, dokonujemy jej zamiany na liczbę dziesiętną i wyświetlamy wynik. Pętla zostanie przerwana (świadomie przez użytkownika) po wprowadzeniu pustego łańcucha znaków lub w wyniku błędnej postaci liczby (wystąpi wyjątek). Ewentualne przechwycenie i obsłużenie wyjątku pozostawimy Czytelnikowi.

```
import java.util.Scanner;
/** Zadanie Z22.2 */
public class Z22_2 {

    /* Tu wstaw cztery tablice: rz, ar, lp i err. */

    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Odczytywanie liczb rzymskich");
        String roman;
        do {
            System.out.print("Podaj liczbę rzymską: ");
            roman = input.next();
            if (roman.equals(""))
                break;

            /* Tu wstaw kod zamiany liczb rzymskich na dziesiętne. */

            System.out.println(roman+" = "+tmp);
        } while (true);
    }
}
```

Zadanie 22.3. Z22_3.java

Na podstawie rozwiązania zadania 22.1 możemy zbudować następującą metodę:

```
private static String decToRoman(int n) {
    if (n < 1 || n > 3999)
        throw new ArithmeticException("Liczba poza zakresem: "+n);
    String[] rz = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
        "V", "IV", "I"};
    int[] ar = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
    StringBuilder tmp = new StringBuilder("");
    for(int i = 0; i<13; ++i) {
        while (n >= ar[i]) {
            n -= ar[i];
            tmp.append(rz[i]);
        }
    }
    return tmp.toString();
}
```

Tablice niezbędne do konwersji liczb dziesiętnych na rzymskie zostały umieszczone w kodzie metody. Zatem kod metody jest kompletny i może w tej postaci być przenoszony do dowolnej aplikacji. Działanie metody pokażemy dla kilkunastu liczb wybranych w sposób losowy.

```
/** Zadanie Z22.3 */
public class Z22_3 {

    /* Tu wstaw kod metody decToRoman() */

    public static void main(String args[]) {
        System.out.println("Zapis liczb w systemie rzymskim");
        for(int i = 0; i < 15; ++i) {
            int n = (int)(1+3999*Math.random());
            System.out.println(n+" = "+decToRoman(n));
        }
    }
}
```

Definicje tablic `rz` i `ar` możemy umieścić w kodzie klasy (tak jak w zadaniu 21.1 lub 21.2) i usunąć z kodu metody `decToRoman()`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 22.4. Z22_4.java

Utworzymy klasę `RN` z dwoma polami: `r` typu `String` (do przechowywania symboli rzymskich) oraz `d` typu `int` (do przechowywania wartości dziesiętnej symbolu). Klasę definiujemy z modyfikatorem `final` — po tej klasie nie będzie można dziedziczyć. Pola klasy definiujemy z modyfikatorami `protected` i `final` (do pól będzie dostęp, ale po utworzeniu obiektu nie będzie można zmienić wartości pól). Konstruktor deklarujemy jako metodę prywatną. Poza klasą `RN` konstruktor nie będzie dostępny. Wykorzystamy go tylko do utworzenia trzynastoelementowej tablicy `rn` (statycznej i finalnej), czyli jedynego elementu udostępnionego przez zdefiniowaną klasę.

```
final class RN {
    protected final String r;
    protected final int d;
    private RN(String r, int d) {
        this.r = r; this.d = d;
    };
    static final RN[] rn = {new RN("M", 1000), new RN("CM", 900),
        new RN("D", 500), new RN("CD", 400),  new RN("C", 100),
        new RN("XC", 90), new RN("L", 50), new RN("XL", 40),
        new RN("X", 10), new RN("IX", 9), new RN("V", 5),
        new RN("IV", 4), new RN("I", 1)};
}
```

Z tej tablicy może korzystać metoda `decToRoman()`:

```
private static String decToRoman(int n) {
    StringBuilder tmp = new StringBuilder("");
    for(int i = 0; i<13; ++i) {
        while (n >= RN.rn[i].d) {
            n -= RN.rn[i].d;
            tmp.append(RN.rn[i].r);
        }
    }
    return tmp.toString();
}
```

Pokazując działanie metody, wyświetlimy potęgi liczby 2 zapisane w systemie rzymskim.

```
/** Zadanie Z22.4 */

    /* Tu wstaw kod klasy RN.*/

public class Z22_4 {

    /* Tu wstaw kod metody decToRoman(). */

    public static void main(String args[]) {
        System.out.println("Potęgi liczby 2 w systemie rzymskim");
        int i = 0, n = 1;
        while (i < 13) {
            System.out.println("2^"+i+" = "+decToRoman(n));
            ++i;
            n *= 2;
        }
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl


Uwaga

Kody obu klas (RN i Z22_4) znajdowały się w jednym pliku: *Z22_4.java*. Po kompilacji otrzymujemy dwa pliki: *RN.class* i *Z22_4.class* — każda skompilowana klasa znajduje się w odrębnym pliku.

Zadanie 22.5. Z22_5.java

Wykorzystajmy klasę RN i metodę decToRoman() z rozwiązania zadania 22.4. Na podstawie rozwiązania zadania 22.2 możemy zbudować metodę:

```
private static int romanToDec(String s) {
    int tmp = 0;
    int index = 0;  // indeks analizowanego znaku
    for (int i = 0; i < 13; i++)
        while (s.startsWith(RS.rn[i].r, index)) {
            tmp += RS.rn[i].d;
            index += RS.rn[i].r.length();
        }
    if (!s.equals(decToRoman(tmp)) || tmp > 3999)
        throw new ArithmeticException("Niewłaściwa liczba rzymska: "+s);
    return tmp;
}
```

Działanie metody sprawdzimy dla kilku liczb rzymskich zapisanych w tablicy.

```
/** Zadanie Z22.5 */

    /* Tu wstaw kod klasy RN lub w bieżącym folderze umieść skompilowany
       plik klasy (RN.class). */

public class Z22_5 {

    /* Tu wstaw kod metody decToRoman(). */
    /* Tu wstaw kod metody romanToDec(). */

    public static void main(String args[]) {
        System.out.println("Odczytywanie liczb w systemie rzymskim");
        String[] test = {"XXVII", "CCXLV", "III", "MMMCMXCIX", "MXXIV"};
        for(String r: test)
            System.out.println(r+" = "+romanToDec(r));
    }
}
```

Zadanie 22.6. Z22_6.java, Roman.java

Na początek tworzymy klasę z jednym prywatnym polem (int n;) i publicznym konstruktorem z jednym parametrem — liczbą całkowitą. Jeśli podana liczba nie mieści się w zakresie od 1 do 3999, to wygenerowany zostanie wyjątek.

```
public class Roman {
    private int n;

    public Roman(int n) {
        if(n < 1 || n > 3999)
            throw new ArithmeticException("Liczba poza zakresem");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        else
            this.n = n;
    }
}
```

Klasę możemy rozszerzyć o kolejne metody — metodę zwracającą wartość dziesiętną liczby naturalnej przechowywanej w obiekcie:

```
public int intValue() {
    return n;
}
```

oraz metodę statyczną zwracającą obiekt klasy `Roman` odpowiadający podanej liczbie całkowitej:

```
public static Roman valueOf(int n) {
    return new Roman(n);
}
```

Z klasy w tej postaci jest jeszcze niewielki pożytek. Do jej kodu dodamy wewnętrzną klasę `RN` (z rozwiązania zadania 22.4), dopisując w nagłówku modyfikator `static`:

```
private static final class RN {
    final String r;
    final int d;
    private RN(String r, int d) {
        this.r = r; this.d = d;
    };
    static final RN[] rn = {new RN("M", 1000), new RN("CM", 900),
        new RN("D", 500), new RN("CD", 400),  new RN("C", 100),
        new RN("XC", 90), new RN("L", 50), new RN("XL", 40),
        new RN("X", 10), new RN("IX", 9), new RN("V", 5),
        new RN("IV", 4), new RN("I", 1)};
}
```

oraz metody statyczne korzystające z tablicy `rn` zdefiniowanej w tej klasie (tym razem zadeklarowane jako publiczne): `decToRoman()` (zadanie 22.4) i `romanToDec()` (zadanie 22.5).

```
public static String decToRoman(int n) {
    StringBuilder tmp = new StringBuilder("");
    for(int i = 0; i<13; ++i) {
        while (n >= RN.rn[i].d) {
            n -= RN.rn[i].d;
            tmp.append(RN.rn[i].r);
        }
    }
    return tmp.toString();
}

public static int romanToDec(String s)  {
    int tmp = 0;
    int index = 0;  // indeks analizowanego znaku
    for (int i = 0; i < 13; i++)
        while (s.startsWith(RN.rn[i].r, index)) {
            tmp += RN.rn[i].d;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            index += RN.rn[i].r.length();
        }
    if (!s.equals(decToRoman(tmp)) || tmp > 3999)
        throw new ArithmeticException("Niewłaściwa liczba rzymska: "
            +s);
    return tmp;
}
```

Korzystając z metody `decToRoman()`, możemy przesłonić metodę `toString()` dziedziczoną z klasy `Object`:

```
@Override public String toString() {
    return decToRoman(this.n);
}
```

Natomiast metodę `romanToDec()` możemy zastosować do zbudowania konstruktora tworzącego nowy obiekt na podstawie liczby zapisanej w systemie rzymskim lub statycznej metody nadającej obiektowi nową wartość utworzoną na podstawie łańcucha znaków.

```
public Roman(String s) {
    this.n = romanToDec(s);;
}

public static Roman valueOf(String s) {
    return new Roman(s);
}
```

Teraz możemy uznać klasę `Roman` za kompletną.


Uwaga

> Klasa `RN` jest klasą wewnętrzną klasy `Roman` i jej kod źródłowy znajduje się w pliku *Roman.java*. Po kompilacji otrzymujemy dwa pliki: *Roman.class* i *Roman$RN.class* — każda skompilowana klasa znajduje się w odrębnym pliku. Klasa `RN` jest dostępna jednak wyłącznie wewnątrz klasy `Roman`.

Przykład wykorzystania obiektów i metod klasy `Roman`:

```
/** Zadanie Z22.6 */
public class Z22_6 {
    public static void main(String args[]) {
        Roman a = new Roman(2012);
        System.out.println(a.intValue()+" = "+a);
        Roman b = Roman.valueOf("XLIV");
        System.out.println(b.intValue()+" = "+b);
        a = Roman.valueOf(137);
        System.out.println(a.intValue()+" = "+a);
        Roman c = new Roman("MMMCMXCIX");
        System.out.println(c.intValue()+" = "+c);
        System.out.println("175 = "+Roman.decToRoman(175));
        System.out.println("XXXIV = "+Roman.romanToDec("XXXIV"));
        System.out.println("MMMM = "+Roman.romanToDec("MMMM"));
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 22.7. Z22_7.java

Ustalimy liczbę pytań (`int pytania = 10`) i maksymalną wartość wylosowanej liczby (`int zakres = 50`). Na podstawie tych parametrów utworzymy tablicę `a`, wypełnioną wylosowanymi i niepowtarzającymi się wartościami. W tym celu wykorzystamy metodę `rndUniqueArray()` z klasy `MyRandomArray`. Wystarczy, że do bieżącego folderu skopiujemy skompilowany kod klasy (*MyRandomArray.class*) lub kod źródłowy klasy (*MyRandomArray.java*). Następnie wartości elementów tablicy zwiększymy o 1; w ten sposób otrzymamy liczby z zakresu od `1` do `50` (zmienna `zakres`).

Ustalimy liczbę punktów (`int pkt = 0`). W pętli zamieniamy liczbę z tablicy na obiekt klasy `Roman`. Wypisujemy liczbę rzymską w konsoli. Użytkownik podaje wartość dziesiętną tej liczby. Jeśli podana wartość jest równa liczbie rzymskiej (`n == roman.intValue()`), powiększamy liczbę punktów (`++pkt`). Po zakończeniu pętli wyświetlimy wynik — sumę zdobytych punktów.

```
import java.util.Scanner;
/** Zadanie Z22.7 */
public class Z22_7 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        int pytania = 10; // 10 pytań
        int zakres = 50;  // największa liczba 50
        int[] a = MyRandomArray.rndUniqueArray(pytania, zakres);
        MyRandomArray.addToArray(a, 1);
        int pkt = 0;
        Roman roman;
        for(int i = 0; i < pytania; ++i) {
            roman = Roman.valueOf(a[i]);
            System.out.print(roman+" = ");
            int n = input.nextInt();
            if (n == roman.intValue()) ++pkt;
        }
        System.out.println("Wynik: "+pkt+" z "+pytania+".");
    }
}
```

Zadanie 22.8. Z22_8.java

Rozwiązanie jest podobne do rozwiązania zadania 22.7. Różnica polega na odwróceniu czynności w głównej pętli programu. Wyświetlamy wartość dziesiętną liczby, a użytkownik wprowadza rzymską postać tej liczby. Do porównania łańcuchów znaków — wprowadzonego przez użytkownika (`s`) i przedstawiającego liczbę rzymską (`roman.toString()`) — użyjemy metody `equals()` z klasy `String`.

```
int pkt = 0;
Roman roman;
for(int i = 0; i < pytania; ++i) {
    roman = Roman.valueOf(a[i]);
    System.out.print(a[i]+" = ");
    String s = input.next();
    if (s.equals(roman.toString())) ++pkt;
}
System.out.println("Wynik: "+pkt+" z 10.");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

23. Trójmian kwadratowy i klasa QuadratPoly

Zadanie 23.1. Z23_1.java, QuadratPoly.java

W klasie definiujemy trzy prywatne pola `a`, `b` i `c` typu `int`, odpowiadające tradycyjnemu zapisowi postaci trójmianu kwadratowego.

Konstruktor z trzema parametrami (`a`, `b` i `c`) pozwala na tworzenie obiektów klasy `QuadratPoly`. Podanie wartości *0* jako pierwszego parametru spowoduje zgłoszenie wyjątku. Metody `value()` obliczające wartość trójmianu dla podanego argumentu definiujemy w dwóch wariantach — dla argumentu całkowitego `n` lub dla zmiennoprzecinkowego argumentu `x`. Wartość trójmianu liczymy według wzoru:

$$ax^2 + bx + c = a(x + b)x + c$$

(tzw. *schemat Hornera* umożliwiający szybkie obliczanie wartości wielomianu dowolnego stopnia).

```
public class QuadratPoly {
    private int a, b, c;

    public QuadratPoly() {}
    public QuadratPoly(int a, int b, int c) {
        if (a == 0)
            throw new ArithmeticException("Niedozwolony parametr
                a = 0!");
        this.a = a;
        this.b = b;
        this.c = c;
    }
    public int value(int n) {
        return (a*n+b)*n+c;
    }
    public double value(double x) {
        return (a*x+b)*x+c;
    }
}
```

W przykładzie demonstrujemy obliczanie wartości trójmianu kwadratowego $2x^2 - 3x + 1$ dla całkowitych argumentów z przedziału $\langle -5, 5 \rangle$ oraz próbę skonstruowania obiektu `QuadratPoly` z niedozwoloną wartością parametru ($a = 0$).

```
/** Zadanie Z23.1 */
public class Z23_1 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -3, 1);
        for(int i = -5; i < 6; ++i)
            System.out.printf("w(%d) = %d\n", i, w.value(i));
        QuadratPoly v = new QuadratPoly(0, 3, -1);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 23.2. Z23_2.java, QuadratPoly.java

Ważnym pojęciem charakteryzującym trójmian kwadratowy jest wyróżnik $\Delta = b^2 - 4ac$. Aby tej wielkości nie obliczać wielokrotnie, dodamy do klasy pole:

```
private int delta;
```

W konstruktorze obliczymy wartość wyróżnika i przechowamy ją w polu `delta`.

```
this.delta = b*b-4*a*c;
```

Tworzymy publiczną metodę zwracającą wartość wyróżnika (`getDelta()`).

```
public int getDelta() {
    return delta;
}
```

W przykładzie utworzymy dwa obiekty `QuadratPoly` (trójmiany kwadratowe) i obliczymy ich wyróżniki.

```
/** Zadanie Z23.2 */
public class Z23_2 {
    public static void main(String args[]) {
        QuadratPoly w1 = new QuadratPoly(2, -3, 1);
        System.out.println("Delta: "+w1.getDelta());
        QuadratPoly w2 = new QuadratPoly(1, 2, 1);
        System.out.println("Delta: "+w2.getDelta());
    }
}
```

Zadanie 23.3. Z23_3.java, QuadratPoly.java

Metoda `toString()` zdefiniowana w klasie `QuadratPoly` przesłoni metodę dziedziczoną z klasy `Object`. Zwrócony łańcuch znaków będzie znanym z matematyki zapisem trójmianu.

Współczynnik *a* jest różny od zera, dla *a = –1* na początku łańcucha umieścimy tylko znak liczby (–), dla *a = 1* współczynnik pominiemy, w pozostałych przypadkach wypisujemy wartość współczynnika. Symbol x^2 w trybie tekstowym zapiszemy w sposób umowny: *x^2*.

Drugi człon trójmianu wypisujemy tylko wtedy, gdy współczynnik *b* nie jest zerem. Dla wartości dodatnich wypisujemy znak +, wartość współczynnika *b* i symbol *x*; dla wartości ujemnych wypisujemy tylko wartość współczynnika *b* i symbol *x*. Podobnie postępujemy w przypadku współczynnika *c*.

```
@Override public String toString() {
    StringBuilder wz = new StringBuilder();
    if (a == -1)
        wz.append("-");
    else if (a != 1)
        wz.append(a);
    wz.append("x^2");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        if (b == -1)
            wz.append("-x");
        else if (b == 1)
            wz.append("+x");
        else if (b > 1)
            wz.append("+").append(b).append("x");
        else if (b < -1)
            wz.append(b).append("x");

        if (c > 0)
            wz.append("+").append(c);
        else if (c < 0)
            wz.append(c);
        return wz.toString();
    }
```

Działanie metody sprawdzimy dla dwóch przykładowych trójmianów. Zauważmy, że w wyrażeniu `"w(x) = "+w` lub `v+" = 0"` metoda jest wywoływana przez operator konkatenacji (`+`). Nie musimy w tej sytuacji jawnie wywoływać metody (np. `"w(x) = "+w.toString()`).

```
/** Zadanie Z23.3 */
public class Z23_3 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -3, 1);
        System.out.println("w(x) = "+w);
        QuadratPoly v = new QuadratPoly(1, 0, -4);
        System.out.println(v+" = 0");
    }
}
```

Zadanie 23.4. Z23_4.java, QuadratPoly.java

Przydatną informacją jest znak wyróżnika trójmianu kwadratowego. Do klasy dodamy metodę zwracającą znak wyróżnika; zrobimy to według następującego schematu: *–1* — `delta` ujemna, *0* — `delta` równa zero, *1* — `delta` dodatnia.

```
public int sgnDelta() {
    return (delta > 0)?1:(delta < 0)?-1:0;
}
```

Współczynniki wielomianu wprowadzimy z klawiatury, korzystając z obiektu `input` klasy `Scanner`. Zwróćmy uwagę na sposób wczytywania współczynnika *a*. Wczytywanie powtarzamy do skutku, aż podana wartość będzie różna od zera. Nie jest to potrzebne w przypadku wczytywania współczynników *b* i *c*.

Jeśli wyróżnik (`delta`) jest ujemny, to równanie nie ma pierwiastków rzeczywistych (*0* pierwiastków). Gdy wyróżnik jest równy zero, mamy jeden pierwiastek rzeczywisty (podwójny). Dla wyróżnika dodatniego są dwa różne pierwiastki rzeczywiste.

Możemy zauważyć związek pomiędzy liczbą pierwiastków i znakiem wyróżnika wyrażonym w postaci liczby (liczba pierwiastków jest równa `1+w.sgnDelta()`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
/** Zadanie Z23.4 */
public class Z23_4 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Podaj współczynniki trójmianu kwadratowego
            ax^2+bx+c");
        int a, b, c;
        do {
            System.out.print("a = ");
            a = input.nextInt();
        } while (a == 0);
        System.out.print("b = ");
        b = input.nextInt();
        System.out.print("c = ");
        c = input.nextInt();
        QuadratPoly w = new QuadratPoly(a, b, c);
        System.out.println("Trójmian kwadratowy: "+w);
        System.out.println("Liczba rzeczywistych pierwiastków: "+
            (1+w.sgnDelta()));
    }
}
```

Odpowiedź możemy sformułować, używając instrukcji warunkowej:

```
System.out.print("Trójmian kwadratowy: "+w+" ");
        int sgn = w.sgnDelta();
        if (sgn == 1)
            System.out.println("ma dwa różne pierwiastki rzeczywiste.");
        else if (sgn == 0)
            System.out.println("ma jeden pierwiastek rzeczywisty
                dwukrotny.");
        else
            System.out.println("nie ma pierwiastków rzeczywistych.");
```

lub wyrażenia warunkowego:

```
int znak = w.sgnDelta();
System.out.println("Liczba rzeczywistych pierwiastków: "+
    (znak == -1)?0:(znak == 0)?:1:2);
```

Kolejną możliwość utworzenia odpowiedzi stwarza nam instrukcja wyboru.

```
String s = "";
switch (w.sgnDelta()) {
    case 1:
        s = "ma dwa różne pierwiastki rzeczywiste.";
        break;
    case 0:
        s = "ma jeden pierwiastek rzeczywisty dwukrotny.";
        break;
    case -1:
        s = "nie ma pierwiastków rzeczywistych.";
        break;
}
System.out.print("Trójmian kwadratowy: "+w+" "+s);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 23.5. Z23_5.java, QuadratPoly.java

Na podstawie wzorów $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ i $x_2 = \frac{-b+\sqrt{\Delta}}{2a}$ budujemy dwie metody (`getX1()` i `getX2()`) zwracające wartości rzeczywistych pierwiastków trójmianu kwadratowego, o ile takie wartości istnieją ($\Delta \geq 0$). Jeśli wyróżnik trójmianu jest ujemny ($\Delta < 0$), to metoda `Math.sqrt(delta)` zwraca wartość `NaN`. Taką samą wartość zwracają metody `getX1()` i `getX2()`.

```
public double getX1() {
    return (-b-Math.sqrt(delta))/(2*a);
}

public double getX2() {
    return (-b+Math.sqrt(delta))/(2*a);
}
```

Współczynniki równania wprowadzimy, stosując obiekt `input` klasy `Scanner`. Wprowadzanie danych można dodatkowo zabezpieczyć przed błędami, przechwytując i obsługując wyjątki pochodzące od metody `nextInt()`. Pozostawimy to do decyzji Czytelnika. W rozwiązaniu zadania wykorzystano metodę `sgnDelta()` i instrukcję wyboru `switch` (zob. objaśnienia z rozwiązania zadania 23.4).

```
import java.util.Scanner;
/** Zadanie Z23.5 */
public class Z23_5 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Podaj współczynniki równania ax^2+bx+c = 0");
        int a, b, c;
        do {
            System.out.print("a = ");
            a = input.nextInt();
        } while (a == 0);
        System.out.print("b = ");
        b = input.nextInt();
        System.out.print("c = ");
        c = input.nextInt();
        QuadratPoly w = new QuadratPoly(a, b, c);
        System.out.print("Równanie "+w+" = 0 ");
        switch (w.sgnDelta()) {
            case 1:
                System.out.println("ma dwa pierwiastki rzeczywiste:");
                System.out.println("x = "+w.getX1());
                System.out.println("x = "+w.getX2());
            break;
            case 0:
                System.out.println("ma pierwiastek rzeczywisty dwukrotny:");
                System.out.println("x = "+w.getX1());
            break;
            case -1:
                System.out.println("nie ma pierwiastków rzeczywistych.");
            break;
        }
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 23.6. Z23_6.java, QuadratPoly.java

Wykresem trójmianu kwadratowego $ax^2 + bx + c, \quad a \neq 0$ jest parabola o wierzchołku $\left(-\frac{b}{2a}, -\frac{\Delta}{4a}\right)$. Metoda `getP()` zwraca pierwszą współrzędną (x) wierzchołka tej paraboli, a metoda `getQ()` — drugą współrzędną (y).

```
public double getP() {
    return -(double)b/(2*a);
}

public double getQ() {
    return -(double)delta/(4*a);
}
```

Pierwsze z powyższych zdań, opisujących to rozwiązanie, formułujemy w programie przykładowym dla konkretnego trójmianu, wykorzystując informacje zwrócone przez metody `toString()`, `getP()` i `getQ()`.

```
/** Zadanie Z23.6 */
public class Z23_6 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -3, 1);
        StringBuilder s = new StringBuilder("Wykresem trójmianu ");
        s.append(w).append(" jest parabola o wierzchołku (");
        s.append(w.getP()).append(", ").append(w.getQ()).append(").");
        System.out.println(s);
    }
}
```

Zadanie 23.7. Z23_7.java, QuadratPoly.java

Użytkownik nie ma dostępu do prywatnych pól klasy. W wielu sytuacjach potrzebuje wiedzieć, jaki znak ma współczynnik `a` trójmianu. Metoda `isAPositive()` zwraca wartość `true`, gdy współczynnik `a` jest dodatni, i wartość `false` w przeciwnym wypadku (wartość wyrażenia logicznego `a > 0`).

```
public boolean isAPositive() {
    return a > 0;
}
```

Jeśli a jest dodatnie, to zbiorem wartości trójmianu $ax^2 + bx + c$ jest przedział $\langle q, +\infty)$, a dla ujemnego współczynnika a — przedział $(-\infty, q\rangle$, gdzie $q = -\frac{\Delta}{4a}$.

```
/** Zadanie Z23.7 */
public class Z23_7 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(-2, -3, 1);
        StringBuilder s = new StringBuilder();
        if (w.isAPositive())
             s.append("<").append(w.getQ()).append(", +oo)");
        else
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            s.append("(-oo, ").append(w.getQ()).append(">");
        System.out.println("Zbiór wartości trójmianu "+w+": "+s);
    }
}
```

Zadanie 23.8. Z23_8.java

Wprowadzenie danych zorganizujemy tak jak w rozwiązaniu zadania 23.4. Jeśli *a* jest dodatnie, to trójmian ax^2+bx+c osiąga w punkcie $x=-\frac{b}{2a}$ wartość minimalną $y_{\min}=-\frac{\Delta}{4a}$, a dla ujemnego współczynnika *a* — wartość maksymalną $y_{\max}=-\frac{\Delta}{4a}$. Budując odpowiedź, użyjemy wartości zwracanych przez metody `getP()` ($-\frac{b}{2a}$) i `getQ()` ($-\frac{\Delta}{4a}$) oraz wyrażenia warunkowego zwracającego odpowiednie słowo (zależne od znaku parametru *a*): `(w.isAPositive())?"minimalną":"maksymalną"`. Odpowiedź budujemy z wykorzystaniem obiektu `s` i metod klasy `StringBuilder`.

```
QuadratPoly w = new QuadratPoly(a, b, c);
StringBuilder s = new StringBuilder("Trójmian kwadratowy ");
s.append(w).append(" osiąga dla x = ").append(w.getP());
s.append(" wartość ").append((w.isAPositive())?"minimalną":"maksymalną");
s.append(" y = ").append(w.getQ()).append(".");
System.out.println(s);
```

Zadanie 23.9. Z23_9.java, QuadratPoly.java

Metoda `getVertex()` zwraca współrzędne wierzchołka paraboli w postaci łańcucha znaków, np. *(–2, 1.5)*.

```
public String getVertex() {
    return "("+getP()+", "+getQ()+")";
}
```

Wynik zwracany przez metodę może być wykorzystany podczas konstruowania opisów właściwości trójmianu.

```
/** Zadanie Z23.9 */
public class Z23_9 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -5, 1);
        System.out.println("Wierzchołek paraboli y = "+w+
            " ma współrzędne "+w.getVertex()+".");
    }
}
```

Zadanie 23.10. Z23_10.java, QuadratPoly.java

W rozwiązaniu zadania 23.7 można było zobaczyć sposób przedstawiania zbioru wartości trójmianu kwadratowego. Na tej podstawie budujemy metodę zwracającą w postaci łańcucha znaków przedział liczbowy będący zbiorem wartości funkcji (przeciwdziedziną).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public String getCodomain() {
    StringBuilder s = new StringBuilder();
        if (isAPositive())
            s.append("<").append(getQ()).append(", +oo)");
        else
            s.append("(-oo, ").append(getQ()).append(">");
    return s.toString();
}
```

Wynik zwracany przez metodę może być wykorzystany podczas opisywania właściwości trójmianu kwadratowego.

```
/** Zadanie Z23.10 */
public class Z23_10 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(-2, -3, 1);
        System.out.println("Zbiór wartości trójmianu "+w+
            ": "+w.getCodomain());
        w = new QuadratPoly(2, 3, -5);
        System.out.println("Zbiór wartości trójmianu "+w+
            ": "+w.getCodomain());
    }
}
```

Zadanie 23.11. Z23_11.java

Korzystając z metod klasy `QuadratPoly` i rozwiązań poprzednich zadań, możemy sporządzić kompletny opis funkcji kwadratowej.

```
/** Zadanie Z23.11 */
public class Z23_11 {
    private static void opisFunkcji(QuadratPoly w) {
        System.out.println("Funkcja kwadratowa f(x) = "+w);
        System.out.println("1. Dziedzina: R (zbiór liczb
            rzeczywistych)");
        System.out.println("2. Zbiór wartości: "+w.getCodomain());
        System.out.println("3. Przedziały monotoniczności:");
        StringBuilder s = new StringBuilder();
        s.append("   - funkcja jest ");
        s.append((w.isAPositive())?"malejąca":"rosnąca");
        s.append(" w przedziale (-oo, ");
        s.append(w.getP()).append(">\n   - funkcja jest ");
        s.append((w.isAPositive())?"rosnąca":"malejąca");
        s.append(" w przedziale <");
        s.append(w.getP()).append(", +oo)");
        System.out.println(s);
        s.delete(0, s.length());
        s.append("4. Ekstremum:\n");
        s.append("   - funkcja osiąga dla x = ").append(w.getP());
        s.append(" wartość ");
        s.append((w.isAPositive())?"minimalną":"maksymalną");
        s.append(" y = ").append(w.getQ());
        System.out.println(s);
        System.out.println("5. Miejsca zerowe:");
        s.delete(0, s.length());
        switch (w.sgnDelta()) {
            case 1:
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                s.append("ma dwa pierwiastki rzeczywiste");
                s.append("\n   x = ").append(w.getX1());
                s.append("\n   x = ").append(w.getX2());
            break;
            case 0:
                s.append("ma pierwiastek rzeczywisty dwukrotny");
                s.append("\n   x = "+w.getX1());
            break;
            case -1:
                s.append("nie ma pierwiastków rzeczywistych");
            break;
        }
        s.insert(0, "   Funkcja ");
        System.out.println(s);
    }
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(-2, -3, 1);
        opisFunkcji(w);
        System.out.println();
        w = new QuadratPoly(1, -2, 1);
        opisFunkcji(w);
        System.out.println();
        w = new QuadratPoly(1, 0, 1);
        opisFunkcji(w);
        System.out.println();
    }
}
```

Zadanie 23.12. Z23_12.java

Wczytywanie danych (współczynników *a*, *b* i *c*) wykonamy podobnie jak w rozwiązaniu 23.4. Zbiór rozwiązań nierówności kwadratowej $ax^2 + bx + c > 0, \quad a \neq 0$, jest zależny od znaku współczynnika *a*, znaku wyróżnika trójmianu (Δ) i rzeczywistych pierwiastków równania kwadratowego $ax^2 + bx + c = 0$.

	Δ > 0	**Δ = 0**	**Δ < 0**
a > 0	$(-\infty, x_1) \cup (x_2, +\infty)$	$(-\infty, x_0) \cup (x_0, +\infty)$ lub $R \setminus \{x_0\}$	$(-\infty, +\infty)$ lub R
a < 0	(x_2, x_1)	$\varnothing$	$\varnothing$

Zauważmy, że gdy $a > 0$ i $\Delta > 0$, to $x_1 = \frac{-b - \sqrt{\Delta}}{2a} < x_2 = \frac{-b + \sqrt{\Delta}}{2a}$, natomiast dla $a < 0$ kolejność pierwiastków jest odwrócona: $x_2 =< x_1$. W przypadku $\Delta = 0$ otrzymujemy pierwiastek dwukrotny $x_0 = -\frac{b}{2a}$ (według powyższych wzorów: $x_0 = x_1 = x_2$).

```
import java.util.Scanner;
/** Zadanie Z23.12 */
public class Z23_12 {
    public static void main(String args[]) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Nierówność kwadratowa ax^2+bx+c > 0");

        /* Deklaracja i wczytanie współczynników a, b i c */

        QuadratPoly w = new QuadratPoly(a, b, c);
        System.out.println("Nierówność "+w+" > 0 ");
        StringBuilder s = new StringBuilder();
        switch (w.sgnDelta()) {
            case 1:
                if (w.isAPositive()) {
                    s.append("(-oo, ").append(w.getX1());
                    s.append(") lub (");
                    s.append(w.getX2()).append(", +oo)");
                } else {
                    s.append("(").append(w.getX2()).append(", ");
                    s.append(w.getX1()).append(")");
                }
                break;
            case 0:
                if (w.isAPositive())
                    s.append("R\\{").append(w.getX1()).append("}");
                else
                    s.append("{} (zbiór pusty)");
                break;
            case -1:
                if (w.isAPositive())
                    s.append("R (zbiór liczb rzeczywistych)");
                else
                    s.append("{} (zbiór pusty)");
                break;
        }
        System.out.println("Zbiór rozwiązań Z = "+s);
    }
}
```

Zadanie 23.13. Z23_13.java

Podobnie jak w rozwiązaniu zadania 23.12, mamy sześć przypadków wyznaczonych przez znaki współczynnika *a* i wyróżnika Δ. W porównaniu z zadaniem 23.12 do zbioru rozwiązań dołączamy dodatkowo miejsca zerowe trójmianu — w nierówności operator *jest większe od 0* (>) został zastąpiony operatorem *jest większe lub równe* ($\geq$).

	$\Delta > 0$	$\Delta = 0$	$\Delta < 0$
a > 0	$(-\infty, x_1\rangle \cup \langle x_2, +\infty)$	$(-\infty, +\infty)$ lub R	$(-\infty, +\infty)$ lub R
a < 0	$\langle x_2, x_1\rangle$	$\{x_0\}$	$\varnothing$

Zmiany w kodzie dotyczą głównie sposobu budowania zbioru rozwiązań nierówności.

```
System.out.println("Nierówność "+w+" >= 0 ");
StringBuilder s = new StringBuilder();
switch (w.sgnDelta()) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        case 1:
            if (w.isAPositive()) {
                s.append("(-oo, ").append(w.getX1()).append("> lub <");
                s.append(w.getX2()).append(", +oo)");
            } else {
                s.append("<").append(w.getX2()).append(", ");
                s.append(w.getX1()).append(">");
            }
            break;
        case 0:
            if (w.isAPositive())
                s.append("R (zbiór liczb rzeczywistych)");
            else
                s.append("{").append(w.getX1()).append("}");
            break;
        case -1:
            if (w.isAPositive())
                s.append("R (zbiór liczb rzeczywistych)");
            else
                s.append("{} (zbiór pusty)");
            break;
        }
    System.out.println("Zbiór rozwiązań Z = "+s);
```

Zadanie 23.14. Z23_14.java, QuadratPoly.java

Do klasy dołączamy metodę `mult()`, która zwraca nowy obiekt `QuadratPoly`, będący iloczynem obiektu wywołującego tę metodę i liczby `n`.

```
public QuadratPoly mult(int n) {
    if (n == 0)
        throw new ArithmeticException("Niewłaściwy parametr, n = 0");
    return new QuadratPoly(n*this.a, n*this.b, n*this.c);
}
```

Przykład wykorzystania metody:

```
/** Zadanie Z23.14 */
public class Z23_14 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -3, -1);
        System.out.println("w(x) = "+w);
        System.out.println("2·w(x) = "+w.mult(2));
        System.out.println("-w(x) = "+w.mult(-1));
    }
}
```

Zadanie 23.15. Z23_15.java, QuadratPoly.java

Zbiór rzeczywistych rozwiązań równania kwadratowego jest zbiorem dwuelementowym ($\Delta > 0$), zbiorem jednoelementowym ($\Delta = 0$) lub zbiorem pustym ($\Delta < 0$). Do klasy `QuadratPoly` dołączamy metodę:

```
public String solutionQE() {
    StringBuilder s = new StringBuilder("{");
    switch (sgnDelta()) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            case 1:
                s.append(getX1()).append(", ").append(getX2()).append("}");
                break;
            case 0:
                s.append(getX1()).append("}");
                break;
            case -1:
                s.append("}");
                break;
            }
        return s.toString();
    }
```

Działanie metody pokażemy, rozwiązując trzy różne równania kwadratowe:

```
/** Zadanie Z23.15 */
public class Z23_15 {
    public static void main(String args[]) {
        QuadratPoly w = new QuadratPoly(2, -3, -1);
        System.out.println("Równanie kwadratowe "+w+
            " = 0\nZbiór rozwiązań Z ="+w.solutionQE()+"\n");
        w = new QuadratPoly(1, -4, 4);
        System.out.println("Równanie kwadratowe "+w+
            " = 0\nZbiór rozwiązań Z ="+w.solutionQE()+"\n");
        w = new QuadratPoly(4, 1, 3);
        System.out.println("Równanie kwadratowe "+w+
            " = 0\nZbiór rozwiązań Z ="+w.solutionQE()+"\n");
    }
}
```

Zadanie 23.16. Z23_16.java, QuadratPoly.java

Na podstawie rozwiązania 23.12 budujemy w klasie `QuadratPoly` prywatną metodę `solutionQIe1()` zwracającą w postaci łańcucha znaków rozwiązanie nierówności kwadratowej $ax^2 + bx + c > 0, \quad a \neq 0$.

```
private String solutionQIe1() {
    StringBuilder s = new StringBuilder();
    switch (sgnDelta()) {
        case 1:
            if (isAPositive()) {
                s.append("(-oo, ").append(getX1());
                s.append(") lub (");
                s.append(getX2()).append(", +oo)");
            } else {
                 s.append("(").append(getX2()).append(", ");
                 s.append(getX1()).append(")");
            }
            break;
        case 0:
            if (isAPositive())
                s.append("R\\{").append(getX1()).append("}");
            else
                s.append("{}");
            break;
        case -1:
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            if (isAPositive())
                s.append("R");
            else
                s.append("{}");
            break;
        }
        return s.toString();
    }
```

Korzystając z rozwiązania zadania 23.13, budujemy w klasie `QuadratPoly` prywatną metodę `solutionQIe2()` zwracającą w postaci łańcucha znaków rozwiązanie nierówności kwadratowej $ax^2 + bx + c \geq 0, \quad a \neq 0$.

```
    private String solutionQIe2() {
        StringBuilder s = new StringBuilder();
        switch (sgnDelta()) {
            case 1:
                if (isAPositive()) {
                    s.append("(-oo, ").append(getX1()).append("> lub <");
                    s.append(getX2()).append(", +oo)");
                } else {
                    s.append("<").append(getX2()).append(", ");
                    s.append(getX1()).append(">");
                }
                break;
            case 0:
                if (isAPositive())
                    s.append("R");
                else
                    s.append("{").append(getX1()).append("}");
                break;
            case -1:
                if (isAPositive())
                    s.append("R");
                else
                    s.append("{}");
                break;
        }
        return s.toString();
    }
```

Publiczna metoda `solutionQIe()` wykorzystuje metody prywatne `solutionQIe1()` i `solutionQIe2()` do rozwiązywania nierówności kwadratowej, której rodzaj jest określony parametrem `typ`, mogącym przyjąć jedną z czterech wartości: `">"`, `"<"`, `">="` i `"<="`. Nierówności w zależności od typu są przekazywane do odpowiedniej metody prywatnej. W dwóch przypadkach niezbędne jest przekształcenie nierówności na nierówność równoważną przez pomnożenie obu jej stron przez liczbę *–1*.

```
    public String solutionQIe(String typ) {
        if (typ.equals(">"))
            return this.solutionQIe1();
        else if (typ.equals(">="))
            return this.solutionQIe2();
        else if (typ.equals("<"))
            return this.mult(-1).solutionQIe1();
        else if (typ.equals("<="))
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            return this.mult(-1).solutionQIe2();
        else
            throw new IllegalArgumentException("Niewłaściwy parametr "+typ);
    }
```

W przykładowym rozwiązaniu wykorzystamy tablicę `qp` zawierającą kilka trójmianów kwadratowych (różniących się znakiem wyróżnika) oraz tablicę zawierającą wszystkie typy nierówności. Przeglądanie tych tablic w dwóch zagnieżdżonych pętlach `for` pozwala nam na zbudowanie 12 zróżnicowanych przykładów. Tablicę `qp` można rozbudować.

```
/** Zadanie Z23.16 */
public class Z23_16 {
    public static void main(String args[]) {
        QuadratPoly[] qp = {new QuadratPoly(2, -3, -1),
            new QuadratPoly(1, -4, 4), new QuadratPoly(4, 1, 3)};
        String[] ie = {">", ">=", "<", "<="};
        for(String typ: ie)
            for(QuadratPoly w: qp) {
                StringBuilder s = new StringBuilder();
                s.append("Nierówność kwadratowa ").append(w).append(" ");
                s.append(typ).append(" 0\nZbiór rozwiązań Z = ");
                s.append(w.solutionQIe(typ)).append("\n");
                System.out.println(s);
            }
    }
}
```

Zadanie 23.17. Z23_17.java

Aby rozwiązać równanie dwukwadratowe $ax^4 + bx^2 + c = 0, \quad a \neq 0$, podstawiamy $y = x^2$ i otrzymujemy równanie kwadratowe $ay^2 + by + c = 0$. Jeśli to równanie ma nieujemne pierwiastki rzeczywiste *y*, to równanie wyjściowe ma pierwiastki o postaci $x = \pm\sqrt{y}$.

Współczynniki równania wprowadzamy tak jak w rozwiązaniu zadania 23.4. Problemem jest wyświetlenie równania dwukwadratowego. Wyjaśnijmy to na przykładzie. Metoda `toString()` zwraca łańcuch o postaci `"2x^2-3x+1"`, w tym łańcuchu odszukamy i zamienimy `"x^2"` na `"x^4"`, następnie zamienimy `"x"` na `"x^2"`. Po tych zamianach otrzymamy `"2x^4-3x^2+1"`.

```
StringBuilder str = new StringBuilder(w.toString());
int pos1 = str.indexOf("x");
str.replace(pos1+2, pos1+3, "4");
int pos2 = str.lastIndexOf("x");
if (pos2 != pos1)
    str.replace(pos2, pos2+1, "x^2");
str.append(" = 0");
```

Równanie dwukwadratowe może mieć co najwyżej 4 pierwiastki rzeczywiste. Deklarujemy czteroelementową tablicę `x` i licznik pierwiastków `p`. Rozwiązujemy równanie kwadratowe z niewiadomą *y* i jeśli rozwiązanie *y* jest nieujemne, to do tablicy dopisujemy dwa pierwiastki $x = -\sqrt{y}$ i $x = \sqrt{y}$ (`x[p++] = -Math.sqrt(y1); x[p++] =`

Ebookpoint.pl kopia dla: pawher@o2.pl

`Math.sqrt(y1);`). Po wyjściu z instrukcji `switch` zmienna `p` zawiera liczbę pierwiastków rzeczywistych równania dwukwadratowego. Pozostaje jedynie wyświetlić te pierwiastki.

```
import java.util.Scanner;
/** Zadanie Z23.17 */
public class Z23_17 {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        System.out.println("Podaj współczynniki równania ax^4+bx^2+c
            = 0");
        int a, b, c;
        do {
            System.out.print("a = ");
            a = input.nextInt();
        } while (a == 0);
        System.out.print("b = ");
        b = input.nextInt();
        System.out.print("c = ");
        c = input.nextInt();
        QuadratPoly w = new QuadratPoly(a, b, c);
        StringBuilder str = new StringBuilder(w.toString());
        int pos1 = str.indexOf("x");
        str.replace(pos1+2, pos1+3, "4");
        int pos2 = str.lastIndexOf("x");
        if (pos2 != pos1)
            str.replace(pos2, pos2+1, "x^2");
        str.append(" = 0");
        System.out.println("Równanie "+str);
        int p = 0; // liczba pierwiastków rzeczywistych
        double[] x = new double[4];
        switch (w.sgnDelta()) {
            case 1:
                double y1 = w.getX1();
                if (y1 >= 0) {
                    x[p++] = -Math.sqrt(y1);
                    x[p++] = Math.sqrt(y1);
                }
                double y2 = w.getX2();
                if (y2 >= 0) {
                    x[p++] = -Math.sqrt(y2);
                    x[p++] = Math.sqrt(y2);
                }
            break;
            case 0:
                double y = w.getX1();
                if (y >= 0) {
                    x[p++] = -Math.sqrt(y);
                    x[p++] = Math.sqrt(y);
                }
            break;
        }
        if (p == 0)
            System.out.println("Nie ma pierwiastków rzeczywistych");
        else if (p == 2) {
            System.out.println("x1 = "+x[0]);
            System.out.println("x2 = "+x[1]);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        } else {
            System.out.println("x1 = "+x[0]);
            System.out.println("x2 = "+x[1]);
            System.out.println("x3 = "+x[2]);
            System.out.println("x4 = "+x[3]);
        }
    }
}
```

Zadanie 23.18. Z23_18.java, QuadratPoly.java

Do klasy `QuadratPoly` dodajemy metody realizujące dodawanie i odejmowanie trójmianów kwadratowych. Jeśli suma (lub różnica w przypadku odejmowania) współczynników przy x^2 jest równa *0*, to wynik nie jest trójmianem kwadratowym i metoda zgłasza wyjątek.

```
public QuadratPoly add(QuadratPoly w) {
    int s = this.a+w.a;
    if (s == 0)
        throw new ArithmeticException("Wynik nie jest trójmianem kwadratowym");
    return new QuadratPoly(s, this.b+w.b, this.c+w.c);
}

public QuadratPoly sub(QuadratPoly w) {
    int s = this.a-w.a;
    if (s == 0)
        throw new ArithmeticException("Wynik nie jest trójmianem kwadratowym");
    return new QuadratPoly(s, this.b-w.b, this.c-w.c);
}
```

Metody przetestujemy, wykonując działania na dwóch wybranych trójmianach.

```
/** Zadanie Z23.18 */
public class Z23_18 {
    public static void main(String args[]) {
        QuadratPoly w1 = new QuadratPoly(2, -3, 4);
        QuadratPoly w2 = new QuadratPoly(-1, 4, 2);
        System.out.println("w1 = "+w1);
        System.out.println("w2 = "+w2);
        System.out.println("w1+w2 = "+w1.add(w2));
        System.out.println("w2+w1 = "+w2.add(w1));
        System.out.println("w1-w2 = "+w1.sub(w2));
        System.out.println("w2-w1 = "+w2.sub(w1));
        System.out.println("w1+2·w2 = "+w1.add(w2.mult(2)));
    }
}
```

Zadanie 23.19. Z23_19.java, FloatQP.java, DoubleQP.java

Wersje zmiennoprzecinkowe klasy `QuadratPoly` zbudujemy, przekształcając odpowiednio kod źródłowy klasy.

a) Budujemy klasę `FloatQP`:

1. Kopiujemy kod z pliku *QuadratPoly.java* do pliku *FloatQP.java*.

Ebookpoint.pl kopia dla: pawher@o2.pl

2. Usuwamy metodę `public int value(int n) { return (a*n+b)*n+c; }`.
3. Zamieniamy w kodzie `QuadratPoly` na `FloatQP` (nazwa klasy, nazwy konstruktorów i typów obiektowych).
4. Zamieniamy w kodzie typ `int` na typ `float`. Wyjątek stanowi metoda `sgnDelta()`, którą pozostawiamy bez zmian:

```
public int sgnDelta() { return (delta > 0)?1:(delta < 0)?-1:0; }.
```

5. Usuwamy rzutowanie wyniku na typ `double` w metodach `getP()` i `getQ()`.
6. Zamieniamy typ `double` na `float`.
7. Wyniki uzyskane przy użyciu metody `Math.sqrt()` rzutujemy na typ `float`.
8. W metodzie `mult()` możemy zmienić parametr `n` na `x`.
9. Do klasy dodajemy metodę `div()`:

```
public FloatQP div(float x) {
    if (x == 0)
        throw new ArithmeticException("Niewłaściwy parametr, x = 0");
    return new FloatQP(a/x, b/x, c/x);
}
```

b) Budujemy klasę `DoubleQP`:

1. Kopiujemy kod z pliku *QuadratPoly.java* do pliku *DoubleQP.java*.
2. Usuwamy metodę `public int value(int n) { return (a*n+b)*n+c; }`.
3. Zamieniamy w kodzie `QuadratPoly` na `DoubleQP` (nazwa klasy, nazwy konstruktorów i typów obiektowych).
4. Zamieniamy w kodzie typ `int` na typ `double`. Wyjątek stanowi metoda `sgnDelta()`, którą pozostawiamy bez zmian:

```
public int sgnDelta() { return (delta > 0)?1:(delta < 0)?-1:0; }.
```

5. Usuwamy rzutowanie wyniku na typ `double` w metodach `getP()` i `getQ()`.
6. W metodzie `mult()` możemy zmienić parametr `n` na `x`.
7. Do klasy dodajemy metodę `div()`:

```
public DoubleQP div(float x) {
    if (x == 0)
        throw new ArithmeticException("Niewłaściwy parametr, x = 0");
    return new DoubleQP(a/x, b/x, c/x);
}
```

Wykorzystanie klas, działanie konstruktorów i metod pokażemy, rozwiązując przykładowe równanie lub nierówność.

```
/** Zadanie Z23.19 */
public class Z23_19 {
    public static void main(String args[]) {
        FloatQP r1 = new FloatQP(1.0f, -4.5f, 0);
        System.out.println("Równanie "+r1+" = 0");
        System.out.println("Rozwiązanie Z = "+r1.solutionQE());
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        DoubleQP r2 = new DoubleQP(-1.0, 2.7, 5.3);
        System.out.println("Nierówność "+r2+" >= 0");
        System.out.println("Rozwiązanie Z = "+r2.solutionQIe(">="));
    }
}
```

24. Rozwiązania zadań — liczby zespolone

Zadanie 24.1. Z24_1.java, Complex.java

Klasa `Complex` zawiera dwa prywatne pola `re` (część rzeczywista liczby, ang. *real*) i `im` (część urojona liczby, ang. *imaginary*) typu `double`.

```
public class Complex {
    private double re, im;
}
```

Do klasy dołączymy trzy podstawowe konstruktory. Pierwszy, bezparametrowy konstruktor zastępuje konstruktor domyślny i buduje obiekt `Complex` odpowiadający liczbie *0+0i* (`re = 0; im = 0;`).

```
public Complex() {}
```

Drugi konstruktor, z jednym parametrem `x` typu `double`, buduje obiekt (liczbę zespoloną) odpowiadający liczbie rzeczywistej *x* (*x+0i* — z zerową częścią urojoną).

```
public Complex(double x) {
    this.re = x;
    this.im = 0;
}
```

Konstruktor z dwoma parametrami, `x` i `y` typu `double`, buduje obiekt odpowiadający liczbie zespolonej *x+yi*.

```
public Complex(double x, double y) {
    this.re = x;
    this.im = y;
}
```

Metody `setRe()` i `setIm()` wywołane przez obiekt ustawiają część rzeczywistą lub urojoną (stosownie do nazwy metody) liczby zespolonej reprezentowanej przez ten obiekt na wartość podaną jako parametr (`x` typu `double`).

```
public void setRe(double x) {
    this.re = x;
}

public void setIm(double x) {
    this.im = x;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Metoda `toString()`utworzona w klasie `Complex` przesłoni metodę `toString()` dziedziczoną z klasy `Object` i zwróci liczbę zespoloną (reprezentowaną przez obiekt `Complex`) w postaci łańcucha znaków, np. *–2.4+3i*, *0–3i*, *4.5* itp.

```
@Override public String toString() {
        StringBuffer tmp = new StringBuffer();
        tmp.append(re);
        if (im > 0) {
            tmp.append("+");
            tmp.append(im);
            tmp.append("i");
        } else if (im < 0) {
            tmp.append(im);
            tmp.append("i");
        }
        return tmp.toString();
    }
```

W przykładzie pokazującym działanie podstawowych konstruktorów i metod klasy `Complex` niezbędne objaśnienia zawarto w komentarzach.

```
/** Zadanie Z24.1 */
public class Z24_1 {
    public static void main(String[] args) {
        /* Konstruktor domyślny */
        Complex a = new Complex();
        System.out.println("a = "+a);
        /* Zmiana części rzeczywistej liczby zespolonej */
        a.setRe(-7.2);
        System.out.println("a = "+a);
        /* Zmiana części urojonej liczby zespolonej */
        a.setIm(4.5);
        System.out.println("a = "+a);
        /* Liczba rzeczywista */
        Complex b = new Complex(5.3);
        System.out.println("b = "+b);
        /* Liczba zespolona */
        Complex c = new Complex(2, -3);
        System.out.println("c = "+c);
        /* Liczba urojona */
        Complex d = new Complex(0, 4);
        System.out.println("d = "+d);
    }
}
```

Zadanie 24.2. Z24_2.java

Jeśli $\Delta \geq 0$, to równanie ma pierwiastki rzeczywiste, które możemy zapisać w postaci zespolonej *x+0i*. Gdy $\Delta < 0$, równanie kwadratowe $az^2 + bz + c = 0, \quad a \neq 0$, ma dwa pierwiastki zespolone: $z = \frac{-b}{2a} - \frac{\sqrt{-\Delta}}{2a} i$ lub $z = \frac{-b}{2a} + \frac{\sqrt{-\Delta}}{2a} i$.

```
public class Z24_2 {
    public static void main(String[] args) {
        double a = 2.0, b = -1.0, c = 0.5;
        double delta = b*b-4*a*c;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        if (delta < 0) {
            double re = -b/(2*a);
            double im = Math.sqrt(-delta)/(2*a);
            Complex z1 = new Complex(re, -im);
            System.out.println("z1 = "+z1);
            Complex z2 = new Complex(re,  im);
            System.out.println("z2 = "+z2);
        } else if (delta > 0) {
            Complex z1 = new Complex((-b-Math.sqrt(delta))/(2*a));
            System.out.println("z1 = "+z1);
            Complex z2 = new Complex((-b+Math.sqrt(delta))/(2*a));
            System.out.println("z2 = "+z2);
        } else {
            Complex z = new Complex(-b/(2*a));
            System.out.println("Pierwiastek dwukrotny z = "+z);
        }
    }
}
```

Zadanie 24.3. Z24_3.java, Complex.java

Sumę dwóch liczb zespolonych obliczamy ze wzoru $(a+bi)+(c+di)=(a+c)+(b+d)i$.

```
public Complex add(Complex x) {
    return new Complex(this.re+x.re, this.im+x.im);
}
```

Podobnie obliczymy różnicę liczb zespolonych: $(a+bi)-(c+di)=(a-c)+(b-d)i$.

```
public Complex sub(Complex x) {
    return new Complex(this.re-x.re, this.im-x.im);
}
```

Kod metod `add()` i `sub()` dołączamy do klasy `Complex` i sprawdzamy ich działanie, wykonując kilka przykładowych obliczeń.

```
public class Z24_3 {
    public static void main(String[] args) {
        Complex a = new Complex(4.5, 7);
        System.out.println("a = "+a);
        Complex b = new Complex(-2, 3.3);
        System.out.println("b = "+b);
        System.out.println("a+b = "+a.add(b));
        System.out.println("b+a = "+b.add(a));
        System.out.println("a-b = "+a.sub(b));
        System.out.println("a-1 = "+a.sub(new Complex(1)));
    }
}
```

Zadanie 24.4. Z24_4.java, Complex.java

Niech z oznacza liczbę zespoloną $a+bi$. Liczbą przeciwną (ang. *opposite* lub *additive inverse*) do liczby z jest liczba $-z=-a-bi$. Jeśli nie ma konfliktów pomiędzy nazwami zmiennych i pól klasy, to możemy stosować zapis skrócony: `re` zamiast `this.re` oraz `im` zamiast `this.im`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public Complex opp() {
    return new Complex(-re, -im);
}
```

Liczbą sprzężoną (ang. *conjugate*) do liczby zespolonej z nazywamy liczbę $\bar{z} = a - bi$.

```
public Complex conj() {
    return new Complex(re, -im);
}
```

Liczbę odwrotną (ang. *reciprocal* lub *multiplicative inverse*) możemy wyznaczyć ze wzoru $z^{-1} = \frac{1}{z} = \frac{\bar{z}}{z \cdot \bar{z}} = \frac{a - bi}{(a + bi)(a - bi)} = \frac{a - bi}{a^2 + b^2} = \frac{a}{a^2 + b^2} + \frac{-b}{a^2 + b^2} i$.

```
public Complex rec() {
    double sc = re*re+im*im;
    return new Complex(re/sc, -im/sc);
}
```

Kilka prostych przykładów ilustruje działanie zdefiniowanych metod.

```
public class Z24_4 {
    public static void main(String[] args) {
        Complex a = new Complex(2, -3.5);
        System.out.println("a = "+a);
        System.out.println("Liczba przeciwna: "+a.opp());
        System.out.println("a+(-a) = "+a.add(a.opp()));
        System.out.println("Liczba sprzężona: "+a.conj());
        System.out.println("Liczba odwrotna: "+a.rec());
    }
}
```

Zadanie 24.5. Z24_5.java, Complex.java

Mnożenie liczb zespolonych wykonamy na podstawie wzoru:

$$(a + bi)(c + di) = ac + adi + bci + bdi^2 = (ac - bd) + (ad + bc)i .$$

```
public Complex mult(Complex x) {
    return new Complex(re*x.re-im*x.im, re*x.im+im*x.re);
}
```

Dzielenie przez liczbę zespoloną zastąpimy mnożeniem przez jej odwrotność (skorzystamy z metody `rec()` podanej w rozwiązaniu zadania 24.4).

```
public Complex div(Complex x) {
    return this.mult(x.rec());
}
```

Działanie metod zademonstrujemy, obliczając kwadrat liczby zespolonej (`a.mult(a)`) oraz iloraz dwóch liczb zespolonych.

```
public class Z24_5 {
    public static void main(String[] args) {
        Complex a = new Complex(0, 1);
        System.out.println("a = "+a);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("a*a = "+a.mult(a));
        Complex b = new Complex(3, 4);
        System.out.println("b = "+b);
        System.out.println("b/a = "+b.div(a));
    }
}
```

Zadanie 24.6. Z24_6.java, Complex.java

Do klasy `Complex` dodajemy trzy stałe odpowiadające liczbom *0*, *1* i jednostce urojonej *i*.

```
public static final Complex ZERO = new Complex(0);
public static final Complex ONE = new Complex(1);
public static final Complex I = new Complex(0, 1);
```

Wykonując serię obliczeń, pokażemy właściwości zdefiniowanych stałych w arytmetyce liczb zespolonych.

```
public class Z24_6 {
    public static void main(String[] args) {
        System.out.println("Własności stałych 0, 1 oraz i");
        System.out.println("a = "+Complex.ZERO);
        System.out.println("0+1 = "+Complex.ZERO.add(Complex.ONE));
        System.out.println("0+i = "+Complex.ZERO.add(Complex.I));
        Complex b = new Complex(2, -3);
        System.out.println("b = "+b);
        System.out.println("b*0 = "+b.mult(Complex.ZERO));
        System.out.println("b*1 = "+b.mult(Complex.ONE));
        System.out.println("b*i = "+b.mult(Complex.I));
        System.out.println("Potęgi jednostki urojonej i");
        System.out.println("i^1 = "+Complex.I);
        System.out.println("i^2 = "+Complex.I.mult(Complex.I));
        System.out.println("i^3 = "+Complex.I.mult(Complex.I).
            mult(Complex.I));
        System.out.println("i^4 = "+Complex.I.mult(Complex.I).
            mult(Complex.I).mult(Complex.I));
    }
}
```

Zadanie 24.7. Z24_7.java, Complex.java

Moduł (bezwzględną wartość) liczby zespolonej $z = a + bi$ możemy obliczyć ze wzoru $|z| = \sqrt{a^2 + b^2}$.

```
public double abs() {
    return Math.sqrt(re*re+im*im);
}
```

Liczbę zespoloną $z = a+bi$ (w postaci kanonicznej) interpretujemy geometrycznie jako punkt *P* o współrzędnych *(a, b)* w prostokątnym (*kartezjańskim*) układzie współrzędnych. Argument liczby zespolonej (różnej od *0+0i*) jest kątem (najczęściej podawanym w radianach) pomiędzy promieniem *PO* (*O* — środek układu współrzędnych) i osią *OX*. Zależność pomiędzy współrzędnymi (a, b) i argumentem liczby zespolonej $z = a+bi$ przedstawia się następująco:

Ebookpoint.pl kopia dla: pawher@o2.pl

$$Arg\ z = \begin{cases} \arctan\left(\frac{b}{a}\right), & \text{gdy } a > 0 \\ \arctan\left(\frac{b}{a}\right) + \pi, & \text{gdy } a < 0 \text{ i } b \geq 0 \\ \arctan\left(\frac{b}{a}\right) - \pi, & \text{gdy } a < 0 \text{ i } b < 0 \\ \frac{\pi}{2}, & \text{gdy } a = 0 \text{ i } b > 0 \\ -\frac{\pi}{2}, & \text{gdy } a = 0 \text{ i } b > 0 \\ 0, & \text{gdy } a = 0 \text{ i } b = 0 \end{cases}$$

Korzystając z metody `Math.atan()`, stałej `Math.PI` oraz instrukcji warunkowych, możemy odpowiednią metodę zbudować. W klasie `Math` mamy zdefiniowaną metodę `atan2()`, która nam to zadanie ułatwia.

```
public double arg() {
    return Math.atan2(im, re);
}
```

Należy dodać, że pojęcie argumentu nie jest jednoznaczne. Nasza metoda `arg()` wyznacza tzw. *argument główny* liczby zespolonej. Pozostałe argumenty tej liczby różnią się od głównego o wielokrotność 2π. W przykładzie obliczymy moduł i argument dla dwóch wybranych liczb zespolonych *4–3i* i *4+4i*. Argument drugiej z tych liczb podajemy w stopniach.

```
public class Z24_7 {
    public static void main(String[] args) {
        Complex a = new Complex(4, -3);
        System.out.println("a = "+a);
        System.out.println("|a| = "+a.abs());
        System.out.println("Arg(a) = "+a.arg());
        a.setIm(4);
        System.out.println("a = "+a);
        System.out.println("|a| = "+a.abs());
        System.out.println("Arg(a) = "+Math.toDegrees(a.arg())+"°");
    }
}
```

Zadanie 24.8. Z24_8.java, Complex.java

Metody `getRe()` i `getIm()` zwracają wartość pól obiektu, czyli część rzeczywistą (`re`) i część urojoną (`im`) liczby zespolonej reprezentowanej przez obiekt.

```
public double getRe() {
    return re;
}

public double getIm() {
    return im;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Działanie metod pokażemy na przykładzie wybranej liczby zespolonej.

```
public class Z24_8 {
    public static void main(String[] args) {
        Complex a = new Complex(4, -3);
        System.out.println("a = "+a);
        System.out.println("Część rzeczywista liczby a: "+a.getRe());
        System.out.println("Część urojona liczby a: "+a.getIm());
    }
}
```

Zadanie 24.9. Z24_9.java, Complex.java

Wywołanie metody `print()` spowoduje wyświetlenie na standardowym wyjściu wartości obiektu w postaci łańcucha znaków.

```
public void print() {
    System.out.print(this.toString());
}
```

Wiersz kodu `System.out.print(this.toString())` możemy zapisać krócej: `System.out.print(this)` lub `System.out.print(toString())`. Gdy w konsoli zostanie wypisana liczba zespolona, kursor pozostanie w bieżącym wierszu. Jeśli będziemy potrzebowali, aby kursor przeszedł na początek nowego wiersza, to wystarczy użyć drugiej metody:

```
public void println() {
    System.out.println(this.toString());
}
```

Ponieważ niejednokrotnie chcemy poprzedzić wyświetlaną wartość dodatkowym tekstem, to zbudujemy wersje tych metod z parametrem typu `String` (wypisywanym przed liczbą).

```
public void print(String str) {
    System.out.print(str+this);
}

public void println(String str) {
    System.out.println(str+this);
}
```

Dysponując tymi metodami, możemy w programach korzystających z obiektów klasy `Complex` zamiast instrukcji `System.out.println("a = "+a)` pisać `a.println("a = ")` — efekt będzie taki sam.

Dokładniej ilustruje to przykład:

```
public class Z24_9 {
    public static void main(String[] args) {
        Complex a = new Complex(4, -3);
        System.out.println("a = "+a);
        System.out.print("a = ");
        a.println();
        a.println("a = ");
        a.add(a).println("a+a = ");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.10. Z24_10.java, Complex.java

Konstruktor kopiujący odczytuje pola obiektu podanego jako parametr i tworzy nowy obiekt z takimi samymi wartościami pól.

```
public Complex(Complex z) {
    this.re = z.re;
    this.im = z.im;
}
```

Dwie liczby zespolone są równe wtedy i tylko wtedy, gdy równe są ich części rzeczywiste i urojone. Porównanie wartości obiektów możemy ograniczyć wyłącznie do obiektów klasy `Complex`:

```
public boolean equals(Complex z) {
    return re == z.re && im == z.im;
}
```

lub wzorując się na rozwiązaniu zadania 20.18, zbudować metodę przesłaniającą metodę `Object.equals()`.

```
@Override public boolean equals(Object o) {
    if (this == o)
        return true;
    if (o == null || !(o instanceof Complex))
        return false;
    Complex c = (Complex) o;
    if (this.re == c.re && this.im == c.im)
        return true;
    else
        return false;
}
```

Przesłaniając metodę `equals()`, powinniśmy przedefiniować metodę `hashCode()`[2]:

```
@Override public int hashCode() {
    int result = 17;
    long tmp = Double.doubleToLongBits(re);
    result = 31*result+(int)(tmp^(tmp>>>32));
    tmp = Double.doubleToLongBits(im);
    result = 31*result+(int)(tmp^(tmp>>>32));
    return result;
}
```

Przykład przedstawia działanie konstruktora kopiującego i metody `equals()`. Proponujemy Czytelnikowi uruchomienie przykładowego kodu i jego dokładną analizę.

```
public class Z24_10 {
    public static void main(String[] args) {
        System.out.println("Dwa nowe obiekty a i b");
        Complex a = new Complex(2, -3);
        a.println("a = ");
        Complex b = new Complex(a);
        b.println("b = ");
```

[2] Opracowane na podstawie: J. Bloch, *Java. Efektywne programowanie. Wydanie II*, Helion 2009.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.print("Porównanie referencji a == b: ");
        System.out.println(a == b);
        System.out.print("Porównanie obiektów a.equals(b): ");
        System.out.println(a.equals(b));
        System.out.println("Nowy obiekt c i podstawienie c = a");
        Complex c;
        c = a;
        c.println("c = ");
        System.out.print("Porównanie referencji a == c: ");
        System.out.println(a == c);
        System.out.print("Porównanie obiektów a.equals(c): ");
        System.out.println(a.equals(c));
        System.out.println("Zmiana wartości c zmienia wartość a");
        c.setRe(0);
        c.println("c = ");
        a.println("a = ");
        System.out.println("Zmiana wartości b nie zmienia wartości a");
        b.setRe(5);
        b.println("b = ");
        a.println("a = ");
        System.out.println("Nowy obiekt d, kopia obiektu c");
        Complex d = new Complex(c); // konstruktor kopiujący
        c.println("c = ");
        d.println("d = ");
        System.out.print("Porównanie referencji c == d: ");
        System.out.println(c == d);
        System.out.print("Porównanie obiektów c.equals(d): ");
        System.out.println(c.equals(d));
    }
}
```

Zadanie 24.11. Z24_11.java, Complex.java

Zaczniemy od sprawdzenia, czy w ciągu znaków występuje symbol jednostki urojonej (*i*). Jeśli nie ma jednostki urojonej, to wprowadzony łańcuch znaków (`s`) może być liczbą rzeczywistą (`re = Double.parseDouble(s); im = 0;`), np. `"-2.4"`. Błąd podczas parsowania tekstu sygnalizuje, że wprowadzona liczba nie jest poprawna (np. `"2o3"`).

Gdy jednostka urojona *i* występuje w łańcuchu, powinna w nim być ostatnim znakiem. Jeśli tak nie jest, to łańcuch nie jest poprawną liczbą zespoloną (np. `"2+i5"` lub `"3.4-5ix"`).

Pozostał nam jeszcze do rozważenia łańcuch typu „$\pm a \pm bi$”, gdzie znaki *a* i *b* oznaczają ciągi cyfr — liczby całkowite lub dziesiętne. Usuwamy ostatni znak i do analizy pozostaje nam ciąg znaków o postaci „$\pm a \pm b$”.

Możliwe są różne sytuacje opisane w komentarzach do kodu.

```
public Complex(String s) {
    int n = s.indexOf("i");
    if (n == -1) {  // Nie ma jednostki urojonej - liczba rzeczywista.
        re = Double.parseDouble(s);
        im = 0;
    } else {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            // Jest jednostka urojona
            if (n == s.length()-1) {
                // i jest ostatnim znakiem.
                s = s.substring(0, s.length()-1);
                if (s.isEmpty() || s.equals("+")) {// liczba "i" lub "+i"
                    re = 0; im = 1; return;
                }
                if (s.equals("-")) {                // liczba "-i"
                    re =0;
                    im = -1;
                    return;
                }
                int znakPlus = s.lastIndexOf("+");
                if (znakPlus == s.length()-1) {     // liczba "a+i" lub błąd
                    re = Double.parseDouble(s.substring(0, znakPlus));
                    im = 1;
                    return;
                }
                int znakMinus = s.lastIndexOf("-");
                if (znakMinus == s.length()-1) {    // liczba "a-i" lub błąd
                    re = Double.parseDouble(s.substring(0, znakMinus));
                    im = -1;
                    return;
                }
                if (znakPlus == -1 && znakMinus == -1) {// "bi" lub błąd
                    re = 0;
                    im = Double.parseDouble(s);
                    return;
                } else if (znakMinus > 0 || znakPlus > 0) {
                    if (znakMinus > 0) {
                        // liczba o postaci "-2-3i", "+2-3i", "2-3i" lub błąd
                        re = Double.parseDouble(s.substring(0, znakMinus));
                        im = Double.parseDouble(s.substring(znakMinus));
                        return;
                    }
                    if (znakPlus > 0) {
                        // liczba o postaci "-2+3i", "+2+3i", "2+3i" lub błąd
                        re = Double.parseDouble(s.substring(0, znakPlus));
                        im = Double.parseDouble(s.substring(znakPlus));
                        return;
                    }
                }
            } else // i nie jest ostatnim znakiem - błędne dane.
                throw new ArithmeticException("Błędna liczba: "+s);
        }
    }
```

Działanie metody sprawdzimy dla kilku przykładowych ciągów znaków, przedstawiających różne możliwe przypadki.

```
public class Z24_11 {
    public static void main(String[] args) {
        String[] test = {"1", "+2.5", "-3.5", "i", "+i", "-i", "2+3i",
            "-2+3i", "2-3i", "-2+3i", "+2-3i", "+2+3i", "1+i", "2-i",
            "-4e-02-2i", "2+4ix"};
        for(String x: test) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            Complex z = new Complex(x);
            System.out.println("\""+x+"\"\t --> "+z);
        }
    }
}
```

Zadanie 24.12. Z24_12.java, Complex.java

Do zbudowania metody statycznej `parseComplex()` zamieniającej łańcuch znaków na obiekt klasy `Complex` możemy wykorzystać (skopiować) kod z rozwiązania zadania 24.11 lub użyć konstruktora.

```
static public Complex parseComplex(String s) {
    return new Complex(s);
}
```

Działanie metody sprawdzimy dla kilku przykładowych ciągów znaków (zapisanych w tablicy `test`), podobnie jak w rozwiązaniu zadania 24.11.

```
for(String x: test)
    System.out.println("\""+x+"\"\t --> "+Complex.parseComplex(x));
```

Zadanie 24.13. Z24_13.java

Przyjmujemy wartość początkową `p = 1+0i`. Dla wykładnika `n > 0` w pętli powtarzamy obliczenie `p = p.mult(a)`, co odpowiada wyrażeniu $a^{n+1} = a^n \cdot a$. Dla `n < 0` stosujemy wzór $a^n = \frac{1}{a^{-n}} = \left(\frac{1}{a}\right)^{-n}$. Mnożenie przez $\frac{1}{a}$ zastępujemy dzieleniem przez *a*, czyli w pętli powtarzamy działanie `p = p.div(a)`. Pozostaje przypadek wykładnika równego *0* i wtedy zwracamy jako wynik początkową wartość `p = 1+0i`.

```
public class Z24_13 {
    public static Complex power(Complex a, int n) {
        Complex p = Complex.ONE;
        if (n > 0)
            for(int i = 0; i < n; ++i)
                p = p.mult(a);
        else if (n < 0)
            for(int i = 0; i < -n; ++i)
                p = p.div(a);
        return p;
    }
    public static void main(String[] args) {
        Complex x = new Complex(1, -1);
        for(int i = -5; i < 10; ++i) {
            System.out.print("("+x.toString()+")^"+i+" = ");
            power(x, i).println();
        }
    }
}
```

Demonstrując działanie metody, obliczyliśmy kilka potęg liczby *1–i*.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.14. Z24_14.java, Complex.java

Metodę statyczną kopiujemy bezpośrednio z rozwiązania zadania 24.13 do klasy `Complex`:

```
public static Complex power(Complex a, int n) {
    Complex p = Complex.ONE;
    if (n > 0)
        for(int i = 0; i < n; ++i)
            p = p.mult(a);
    else if (n < 0)
        for(int i = 0; i < -n; ++i)
            p = p.div(a);
    return p;
}
```

Kopiujemy kod metody po raz drugi, usuwany słowo kluczowe `static` i parametr `a` z nagłówka funkcji. W ciele metody zmienną `a` zastępujemy słowem kluczowym `this`.

```
public Complex power(int n) {
    Complex p = Complex.ONE;
    if (n > 0)
        for(int i = 0; i < n; ++i)
            p = p.mult(this);
    else if (n < 0)
        for(int i = 0; i < -n; ++i)
            p = p.div(this);
    return p;
}
```

Obliczymy potęgi jednostki urojonej *i* o wykładnikach od *–4* do *9*. Do obliczeń wykorzystamy obie metody.

```
public class Z24_14 {
    public static void main(String[] args) {
        Complex x = new Complex(0, 1);
        for(int i = -4; i < 5; ++i)
            System.out.println("("+x+")^"+i+" = "+Complex.power(x, i));
        for(int i = 5; i < 10; ++i)
            System.out.println("("+x+")^"+i+" = "+x.power(i));
    }
}
```

Zadanie 24.15. Complex.java, DemoComplex_15.java

Zgodnie z definicją $a^2 = a \cdot a$ kwadrat liczby zespolonej reprezentowanej przez obiekt `a` wyznaczymy, wykonując mnożenie `a.mult(a)`.

```
public Complex sqr() {
    return this.mult(this);
}
```

Podobnie możemy zbudować metodę statyczną:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public static Complex sqr(Complex z) {
    return z.mult(z);
}
```

lub:

```
public static Complex sqr(Complex z) {
    return z.sqr();
}
```

Aby obliczyć pierwiastek drugiego stopnia $\sqrt{a+bi}$, rozwiążemy równanie kwadratowe $(x+yi)^2 = a+bi$. Po przekształceniu otrzymamy układ równań: $\begin{cases} x^2 - y^2 = a \\ 2xy = b \end{cases}$.

Z układu równań otrzymamy $x = \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}}$ i $y = \varepsilon\sqrt{\frac{\sqrt{a^2+b^2}-a}{2}}$, gdzie ε oznacza znak części urojonej liczby podpierwiastkowej (znak *b*: –1 lub 1). Stąd mamy dwa pierwiastki $\pm(x+yi)$. Przypadek $b = 0$ należy rozpatrywać osobno. Dla $a \geq 0$ otrzymamy w wyniku liczbę rzeczywistą $\pm\sqrt{a}$, a dla $a < 0$ liczbę urojoną $\pm i\sqrt{-a}$.

```
public Complex sqrt() {
    if (im == 0)
        if (re >= 0)
            return new Complex(Math.sqrt(re));
        else
            return new Complex(0.0, Math.sqrt(-re));
    else
        return new Complex(Math.sqrt((abs()+re)/2),
            Math.signum(im)*Math.sqrt((abs()-re)/2));
}
```

Drugi (następny) pierwiastek jest liczbą przeciwną (do poprzedniego):

```
public Complex nextSqrt() {
    return this.sqrt().opp();
}
```

Po niewielkich poprawkach otrzymamy metody statyczne obliczające pierwiastek drugiego stopnia z liczby zespolonej:

```
public static Complex sqrt(Complex z) {
    if (z.im == 0)
        if (z.re >= 0)
            return new Complex(Math.sqrt(z.re));
        else
            return new Complex(0.0, Math.sqrt(-z.re));
    else
        return new Complex(Math.sqrt((z.abs()+z.re)/2),
            Math.signum(z.im)*Math.sqrt((z.abs()-z.re)/2));
}

public static Complex nextSqrt(Complex z) {
    return sqrt(z).opp();
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

W przykładzie sprawdzamy na wybranych wartościach liczb zespolonych zdefiniowane metody `sqr()` i `sqrt()`, składając te funkcje $\sqrt{a^2}$ lub $\left(\sqrt{a}\right)^2$.

```
public class Z24_15 {
    public static void main(String[] args) {
        /* Podnoszenie pierwiastka kwadratowego do kwadratu */
        new Complex( 1, -2).sqrt().sqr().println();
        new Complex( 1,  2).sqrt().sqr().println();
        new Complex(-1, -2).sqrt().sqr().println();
        new Complex(-1,  2).sqrt().sqr().println();
        /* Pierwiastkowanie kwadratu liczby zespolonej */
        new Complex( 1, -2).sqr().sqrt().println();
        new Complex( 1,  2).sqr().sqrt().println();
        new Complex(-1, -2).sqr().sqrt().println(); // liczba przeciwna
        new Complex(-1, -2).sqr().nextSqrt().println();
        new Complex(-1,  2).sqr().sqrt().println(); // liczba przeciwna
        new Complex(-1,  2).sqr().nextSqrt().println();
        /* Metody statyczne sqr i sqrt */
        Complex x = new Complex(2.5, -4);
        System.out.println("x = "+x);
        Complex y = Complex.sqr(x);
        System.out.println("y = x^2 = "+y);
        System.out.println("z = sqrt(y) = "+Complex.sqrt(y));
        System.out.println("z = sqrt(y) = "+Complex.nextSqrt(y));
    }
}
```

Zadanie 24.16. Z24_16.java, Complex.java

Aby wyświetlić część rzeczywistą i urojoną liczby zespolonej w konsoli, użyjemy metody `System.out.printf()`. Zwróćmy uwagę na specyfikator `"%f%+fi"`. Występujący w łańcuchu znak plus (+) powoduje, że przed drugą liczbą (częścią urojoną) będzie wyświetlany znak liczby — plus również.

```
public void printf() {
    System.out.printf("%f%+fi", this.re, this.im);
}

public void printf(String str) {
    System.out.printf(str+"%f%+fi", this.re, this.im);
}

public void printlnf() {
    System.out.printf("%f%+fi\n", this.re, this.im);
}

public void printlnf(String str) {
    System.out.printf(str+"%f%+fi\n", this.re, this.im);
}
```

Czytelnik może zmodyfikować te metody, dodając parametr określający precyzję (liczbę miejsc po przecinku) lub przekazując specyfikator. W przykładzie porównajmy sposób działania metody `println()` (z zadania 24.9) i zdefiniowanej metody `printf()`.

```
public class Z24_16 {
    public static void main(String[] args) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        Complex a = new Complex(-5);
        a.println("Metoda println: \ta = ");
        a.printlnf("Metoda printlnf:\ta = ");
        a.sqrt().println("Metoda println: \tsqrt(a) = ");
        a.sqrt().printlnf("Metoda printlnf:\tsqrt(a) = ");
    }
}
```

Zadanie 24.17. Z24_17.java

Rozwiązywanie równania kwadratowego o współczynnikach zespolonych w zbiorze liczb zespolonych sprowadza się do obliczenia wyróżnika ($\Delta = b^2 - 4ac$) i wyznaczenia pierwiastków równania na podstawie wzoru $z = \pm\frac{-b-\sqrt{\Delta}}{2a}$.

```
public class Z24_17 {
    public static void main(String[] args) {
        /* Współczynniki zespolone równania az^2+bz+c = 0 */
        Complex a = new Complex(1, -2);
        a.printlnf("a = ");
        Complex b = new Complex(0, 1);
        b.printlnf("b = ");
        Complex c = new Complex(-1);
        c.printlnf("c = ");
        System.out.println("Równanie ("+a+")z^2+("+b+")z+"+a+" = 0");
        /* Obliczenie pierwiastków kwadratowych */
        Complex delta = b.sqr().sub(new Complex(4).mult(a).mult(c));
        delta.printlnf("delta = ");
        Complex z1 = b.opp().sub(delta.sqrt()).div(new Complex(2).
            mult(a));
        z1.printlnf("z1 = ");
        Complex z2 = b.opp().add(delta.sqrt()).div(new Complex(2).
            mult(a));
        z2.printlnf("z2 = ");
        System.out.println("Sprawdzenie:");
        a.mult(z1.sqr()).add(b.mult(z1)).add(c).printlnf("w(z1) = ");
        a.mult(z2.sqr()).add(b.mult(z2)).add(c).printlnf("w(z2) = ");
    }
}
```

Zadanie 24.18.

Oznaczmy $x = a + bi$ oraz $y = c + di$.

Suma: $x + y = (a + bi) + (c + di) = (a + c) + (b + d)i$.

```
public static Complex sum(Complex x, Complex y) {
    return new Complex(x.re+y.re, x.im+y.im);
}
```

Różnica: $x - y = (a + bi) - (c + di) = (a - c) + (b - d)i$.

```
public static Complex diff(Complex x, Complex y) {
    return new Complex(x.re-y.re, x.im-y.im);
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Iloczyn: $x \cdot y = (a+bi)\cdot(c+di) = (ac-bd)+(ad+bc)i$.

```
public static Complex prod(Complex x, Complex y) {
    return new Complex(x.re*y.re-x.im*y.im, x.re*y.im+x.im*y.re);
}
```

Iloraz: $\frac{x}{y} = \frac{a+bi}{c+di} = \frac{(a+bi)\cdot(c-di)}{(c+di)\cdot(c-di)} = \frac{(ac+bd)+(bc-ad)i}{c^2+d^2} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$.

```
public static Complex quot(Complex x, Complex y) {
    double s = x.re*x.re+y.re*y.re;
    return new Complex((x.re*y.re+x.im*y.im)/s, (x.im*y.re-x.re*y.im)/s);
}
```

Jak wykorzystać w programie zdefiniowane metody, pokazuje przykład:

```
public class Z24_18 {
    public static void main(String[] args) {
        Complex a = new Complex(1, -2);
        a.printlnf("a = ");
        Complex b = new Complex(0, 1);
        b.printlnf("b = ");
        Complex c = Complex.sum(a, b);
        c.printlnf("a+b = ");
        c = Complex.diff(a, b);
        c.printlnf("a-b = ");
        c = Complex.prod(a, b);
        c.printlnf("a·b = ");
        c = Complex.quot(a, b);
        c.printlnf("a/b = ");
    }
}
```

Zadanie 24.19. Z24_19.java, PolarComplex.java

Klasa `PolarComplex` zawiera dwa prywatne pola `r` (promień, moduł liczby zespolonej, $r \geq 0$) i `fi` (kąt, argument liczby zespolonej) oraz domyślny konstruktor (nadający obiektowi wartość `0`).

```
public class PolarComplex {
private double r, fi;
    public PolarComplex() {}
}
```

Do klasy dołączmy konstruktor z parametrami `r` i `fi`:

```
public PolarComplex(double r, double fi) {
    if (r < 0)
        throw new ArithmeticException("Niewłaściwy parametr: "+r);
    this.r = r;
    this.fi = fi;
}
```

oraz konstruktor z jednym parametrem `r`, przyjmujący domyślne $\varphi = 0$ (`this.fi = 0.0`) dla liczby rzeczywistej nieujemnej `r`, $\varphi = -\pi$ (`this.fi = -Math.PI`) dla liczby ujemnej `r`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public PolarComplex(double r) {
    this.r = Math.abs(r);
    if (r >= 0.0)
        this.fi = 0.0;
    else
        this.fi = -Math.PI;
}
```

Zamianę współrzędnych biegunowych na kartezjańskie zrealizujemy za pomocą wzoru $\begin{cases} x = r\cos\varphi \\ y = r\sin\varphi \end{cases}$:

```
public Complex toComplex() {
    return new Complex(r*Math.cos(fi), r*Math.sin(fi));
}
```

Wartości pól obiektu (`r` i `fi`) typu `double` zamienimy na łańcuchy znaków reprezentujące liczby z sześcioma miejscami po przecinku, z których następnie budujemy wynik (łańcuch wyjściowy). Do formatowania liczby używamy obiektu `formatter` klasy `DecimalFormat` (dziedziczącej z klasy `java.text.NumberFormat` — niezbędne jest importowanie odpowiednich klas w klasie `PolarComplex`):

```
import java.text.DecimalFormat;
import java.text.NumberFormat;
```

Przesłaniamy metodę `toString()` z klasy `Object`:

```
@Override public String toString() {
    NumberFormat formatter = new DecimalFormat("#0.000000");
    String s1 = formatter.format(this.r);
    String s2 = formatter.format(this.fi);
    return "[r="+s1+"; fi="+s2+"]";
}
```

Działanie konstruktorów i metody `toComplex()` pokażemy dla kilku wybranych wartości.

```
/** Zadanie Z24_19 */
public class Z24_19 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex();
        System.out.println("a = "+a.toComplex()+" = "+a);
        PolarComplex b = new PolarComplex(5.4);
        System.out.println("b = "+b.toComplex()+" = "+b);
        PolarComplex c = new PolarComplex(-1);
        System.out.println("c = "+c.toComplex()+" = "+c);
        c.toComplex().printlnf("c = ");
        PolarComplex d = new PolarComplex(Math.sqrt(2), Math.PI/4);
        System.out.println("d = "+d.toComplex()+" = "+d);
        d.toComplex().printlnf("d = ");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.20. Z24_20.java, PolarComplex.java

Do klasy `PolarComplex` dodajemy konstruktor kopiujący, który tworzy nowy obiekt `PolarComplex`, reprezentujący tę samą wartość zespoloną co obiekt przekazany jako parametr (efektem porównania tych obiektów metodą `equals()` powinna być wartość `true`, natomiast porównanie referencji (`==`) da wynik `false`).

```
public PolarComplex(PolarComplex z) {
    r = z.r;
    fi = z.fi;
}
```

Drugi konstruktor tworzy nowy obiekt `PolarComplex`, reprezentujący liczbę zespoloną podaną jako parametr typu `Complex` ($r = |z|$ i $\varphi = \arg z$).

```
public PolarComplex(Complex z) {
    r = z.abs();
    fi = z.arg();
}
```

W przykładowym programie tworzymy obiekt `a` klasy `PolarComplex`, reprezentujący liczbę zespoloną *i* (jednostkę urojoną) — w postaci kanonicznej *0+i*, w postaci biegunowej $\left(1, \frac{\pi}{2}\right)$. Obiekt *b* jest utworzony przez konstruktor kopiujący z obiektu *a*. Zamieniając te obiekty na obiekty klasy `Complex` (metoda `toComplex()`), zauważymy rozbieżności na dalszych miejscach po przecinku (rzędu 10^{-16}). Podobną precyzję uzyskamy podczas konwersji liczby *3–4i* i innych liczb zespolonych.

```
/** Zadanie Z24.20 */
public class Z24_20 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/2);
        System.out.println("a = "+a.toComplex());
        PolarComplex b = new PolarComplex(a);
        System.out.println("b = "+b.toComplex());
        PolarComplex c = new PolarComplex(new Complex(3, -4));
        System.out.println("c = "+c.toComplex());
    }
}
```

Zadanie 24.21. Z24_21.java, Complex.java

W klasie `Complex` dodajemy konstruktor budujący nowy obiekt `Complex`, odpowiadający liczbie zespolonej przekazanej jako parametr typu `PolarComplex`. Do zmiany typu wykorzystamy metodę `toComplex()` z klasy `PolarComplex` (`Complex tmp = z.toComplex()`). Następnie wartości pól obiektu `tmp` skopiujemy do pól nowego obiektu.

```
public Complex(PolarComplex z) {
    Complex tmp = z.toComplex();
    this.re = tmp.re;
    this.im = tmp.im;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Do konwersji obiektu `Complex` na obiekt `PolarComplex` zbudujemy metodę `toPolarComplex()` ($r = |z|$ i $\varphi = \arg z$).

```
public PolarComplex toPolarComplex() {
    return new PolarComplex(this.abs(), this.arg());
}
```

Podobnie jak w rozwiązaniu zadania 24.20, pokazujemy konwersję pomiędzy obiektami klasy `PolarComplex` i `Complex`.

```
/** Zadanie Z24.21 */
public class Z24_21 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/2);
        System.out.println("a = "+a.toComplex());
        Complex b = new Complex(a);
        System.out.println("b = "+b);
        Complex c = new Complex(3, -4);
        c.toPolarComplex().toComplex().println("c = ");
    }
}
```

Zadanie 24.22. Z24_22.java, PolarComplex.java

Iloczyn dwóch liczb zespolonych zapisanych w postaci trygonometrycznej:

$$x = |x|(\sin \varphi + i \cos \varphi) \text{ i } y = |y|(\sin\psi + i\cos\psi)$$

wyraża się wzorem:

$$xy = |x||y|(\sin(\varphi+\psi) + i\sin(\varphi+\psi))$$

Na tej podstawie możemy zbudować metodę obliczającą iloczyn liczb zespolonych — obiektów klasy `PolarComplex`.

```
public PolarComplex mult(PolarComplex z) {
    return new PolarComplex(this.r*z.r, this.fi+z.fi);
}
```

W przykładzie obliczymy iloczyn dwóch liczb zespolonych podanych w postaci biegunowej (obiektów klasy `PolarComplex`) i wyświetlimy wyniki w postaci kanonicznej (po zamianie wyniku na obiekt `Complex`) z dokładnością do 6 miejsc po przecinku (metoda `printlnf()` z zadania 24.16) oraz w postaci biegunowej (stosując metodę `toString()` dla obiektu klasy `PolarComplex` — wywoływaną podczas łączenia tekstów w parametrze metody, np. `System.out.println("a = "+a)`).

```
/** Zadanie Z24.22 */
public class Z24_22 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/4);
        a.toComplex().printlnf("a = ");
        PolarComplex b = new PolarComplex(-2, Math.PI/2);
        b.toComplex().printlnf("b = ");
        a.mult(b).toComplex().printlnf("a·b = ");
        b.mult(a).toComplex().printlnf("b·a = ");
        System.out.println("a = "+a);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("b = "+b);
        System.out.println("a·b = "+a.mult(b));
    }
}
```

Zadanie 24.23. Z24_23.java, PolarComplex.java

Zapiszmy liczby zespolone w postaci trygonometrycznej:

$$x = |x|(\sin\ \varphi\ +\ i\ \cos\ \varphi\ )\ \text{i}\ y = |y|(\sin\psi + i\cos\psi)$$

Iloraz tych liczb wyraża się wzorem:

$$xy = \frac{|x|}{|y|}(\sin(\varphi - \psi) + i\sin(\varphi - \psi))$$

który wykorzystamy do zbudowania metody obliczającej iloraz liczb zespolonych.

```
public PolarComplex div(PolarComplex z) {
    return new PolarComplex(this.r/z.r, this.fi-z.fi);
}
```

Obliczymy iloraz dwóch liczb i podobnie jak w rozwiązaniu zadania 24.22, wyświetlimy wyniki w postaci kanonicznej oraz w postaci biegunowej.

```
/** Zadanie Z24.23 */
public class Z24_23 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/4);
        a.toComplex().printlnf("a = ");
        PolarComplex b = new PolarComplex(-2, Math.PI/2);
        b.toComplex().printlnf("b = ");
        Complex x = a.div(b).toComplex();
        x.printlnf("x = a/b = ");
        Complex y = b.div(a).toComplex();
        y.printlnf("y = b/a = ");
        x.mult(y).printlnf("x·y = ");
        System.out.println("a = "+a);
        System.out.println("b = "+b);
        System.out.println("a/b = "+a.div(b));
        System.out.println("b/a = "+b.div(a));
    }
}
```

Zadanie 24.24. Z24_24.java, PolarComplex.java

Stosując wzór na iloczyn liczb zespolonych, podany w rozwiązaniu zadania 24.22, obliczymy kwadrat i sześcian liczby zespolonej (podanej w postaci biegunowej).

```
public PolarComplex sqr() {
    return new PolarComplex(r*r, 2*fi);
}

public PolarComplex cube() {
    return new PolarComplex(r*r*r, 3*fi);
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Składając w przykładowym programie odpowiednio metody `sqr()`, `cube()` oraz `mult(a)`, obliczymy potęgi (o wykładnikach od *1* do *6*) liczby zespolonej reprezentowanej przez obiekt `a`. Obliczenia wykonamy dla liczby $\frac{1+i\sqrt{3}}{2}$ (w postaci biegunowej $r=1$, $\varphi=\frac{\pi}{3}rad=60°$).

```
/** Zadanie Z24.24 */
public class Z24_24 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/3);
        a.toComplex().printf("a^1 = ");
        System.out.println(" = "+a);
        a.sqr().toComplex().printf("a^2 = ");
        System.out.println(" = "+a.sqr());
        a.cube().toComplex().printf("a^3 = ");
        System.out.println(" = "+a.cube());
        a.sqr().sqr().toComplex().printf("a^4 = ");
        System.out.println(" = "+a.sqr().sqr());
        a.sqr().sqr().mult(a).toComplex().printf("a^5 = ");
        System.out.println(" = "+a.sqr().sqr().mult(a));
        a.sqr().cube().toComplex().printf("a^6 = ");
        System.out.println(" = "+a.sqr().cube());
    }
}
```

Zadanie 24.25. Z24_25.java, PolarComplex.java

Potęgowanie liczb zapisanych w postaci trygonometrycznej $z=|z|(\cos\varphi+i\sin\varphi)$ (biegunowej) jest łatwe przy wykorzystaniu *wzoru de Moivre'a*: $z^n=|z|^n(\cos n\varphi+i\sin n\varphi)$.

```
public PolarComplex power(int n) {
    return new PolarComplex(Math.pow(r, n), n*fi);
}
```

Aby zilustrować zastosowanie metody `power()`, obliczymy potęgi (o wykładnikach od *0* do *12*) liczby zespolonej reprezentowanej przez obiekt `a`. Obliczenia wykonamy dla liczby $\frac{\sqrt{3}+i}{2}$ (w postaci biegunowej $r=1$, $\varphi=\frac{\pi}{6}rad=30°$.

```
/** Zadanie Z24.25 */
public class Z24_25 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(1, Math.PI/6);
        for(int i = 0; i < 13; ++i) {
            a.power(i).toComplex().printf("a^"+i+" = ");
            System.out.println(" = "+a.power(i));
        }
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.26. Z24_26.java, PolarComplex.java

Niezerowa liczba zespolona z ma n różnych zespolonych pierwiastków n-tego stopnia, które można obliczyć, korzystając ze wzoru de Moivre'a:

$$z_k = \sqrt[n]{|z|}\left(\cos\frac{\varphi+2k\pi}{n} + i\sin\frac{\varphi+2k\pi}{n}\right) \text{ dla } k = 0, 1, \ldots, n-1.$$

Dla $n = 2$ otrzymujemy:

$$z_k = \sqrt{|z|}\left(\cos\frac{\varphi+2k\pi}{2} + i\sin\frac{\varphi+2k\pi}{2}\right) = \sqrt{|z|}\left(\cos\left(\frac{\varphi}{2}+k\pi\right) + i\sin\left(\frac{\varphi}{2}+k\pi\right)\right) \text{ dla } k = 0, 1.$$

```
public PolarComplex sqrt(int k) {
    k %= 2;
    return new PolarComplex(Math.sqrt(r), fi/2+k*Math.PI);
}
```

Jako przykład działania metody obliczymy pierwiastki stopnia drugiego z liczby urojonej $2i$.

```
/** Zadanie Z24.26 */
public class Z24_26 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(2, Math.PI/2);
        System.out.print("a = "+a+" = ");
        a.toComplex().printlnf();
        System.out.println("Pierwiastki drugiego stopnia");
        PolarComplex x1 = a.sqrt(0);
        System.out.print("x1 = "+x1+" = ");
        x1.toComplex().printlnf();
        PolarComplex x2 = a.sqrt(1);
        System.out.print("x2 = "+x2+" = ");
        x2.toComplex().printlnf();
    }
}
```

Zadanie 24.27. Z24_27.java, PolarComplex.java

Na podstawie wzoru de Moivre'a (zob. rozwiązanie zadania 24.26) dla $n = 3$ otrzymujemy $z_k = \sqrt[3]{|z|}\left(\cos\frac{\varphi+2k\pi}{3} + i\sin\frac{\varphi+2k\pi}{3}\right)$ dla $k = 0, 1, 2$.

```
public PolarComplex cbrt(int k) {
    k %= 3;
    return new PolarComplex(Math.cbrt(r), (fi+2*k*Math.PI)/3);
}
```

Podając przykład działania metody, obliczymy pierwiastki stopnia trzeciego z liczby -8 (w tym jeden pierwiastek rzeczywisty -2).

```
/** Zadanie Z24.27 */
public class Z24_27 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(8, -Math.PI);
        System.out.print("a = "+a+" = ");
        a.toComplex().printlnf();
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        System.out.println("Pierwiastki trzeciego stopnia");
        PolarComplex x1 = a.cbrt(0);
        System.out.print("x1 = "+x1+" = ");
        x1.toComplex().printlnf();
        PolarComplex x2 = a.cbrt(1);
        System.out.print("x2 = "+x2+" = ");
        x2.toComplex().printlnf();
        PolarComplex x3 = a.cbrt(2);
        System.out.print("x3 = "+x3+" = ");
        x3.toComplex().printlnf();
    }
}
```

Zadanie 24.28. Z24_28.java, PolarComplex.java

Na podstawie rozwiązania zadania 24.26 budujemy bezparametrową metodę `sqrt()` zwracającą tablicę obiektów `Complex` z dwoma pierwiastkami drugiego stopnia z liczby zespolonej reprezentowanej przez obiekt `PolarComplex` wywołujący metodę.

```
public Complex[] sqrt() {
    Complex[] tmp = new Complex[2];
    tmp[0] = new PolarComplex(Math.sqrt(r), fi/2).toComplex();
    tmp[1] = new PolarComplex(Math.sqrt(r), fi/2+Math.PI).toComplex();
    return tmp;
}
```

Podobnie (zob. rozwiązanie zadania 24.27) budujemy bezparametrową metodę `cbrt()` zwracającą tablicę obiektów `Complex` z trzema pierwiastkami trzeciego stopnia z liczby zespolonej.

```
public Complex[] cbrt() {
    Complex[] tmp = new Complex[3];
    tmp[0] = new PolarComplex(Math.sqrt(r), fi/3).toComplex();
    tmp[1] = new PolarComplex(Math.sqrt(r), (fi+2*Math.PI)/3).
        toComplex();
    tmp[2] = new PolarComplex(Math.sqrt(r), (fi+4*Math.PI)/3).
        toComplex();
    return tmp;
}
```

W przykładowym programie obliczymy pierwiastki drugiego stopnia z *–4* i pierwiastki trzeciego stopnia z *8i*.

```
/** Zadanie Z24.28 */
public class Z24_28 {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(new Complex(4));
        System.out.println("Pierwiastki kwadratowe z liczby 4:");
        for(Complex x: a.sqrt()) {
            x.printf("x = ");
            System.out.println(" = "+x.toPolarComplex());
        }
        a = new PolarComplex(new Complex(-4));
        System.out.println("Pierwiastki kwadratowe z liczby -4:");
        for(Complex x: a.sqrt()) {
            x.printf("x = ");
            System.out.println(" = "+x.toPolarComplex());
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        }
        a = new PolarComplex(new Complex(0, 8));
        System.out.println("Pierwiastki sześcienne z liczby 8i:");
        for(Complex x: a.cbrt()) {
            x.printf("x = ");
            System.out.println(" = "+x.toPolarComplex());
        }
    }
}
```

Ponieważ bezparametrowe metody `sqrt()` i `cbrt()` zwracają pierwiastki zebrane w tablicy, to do wyświetlania wartości możemy użyć statycznej metody `toString()` z klasy `Arrays`.

```
import java.util.Arrays;
/** Zadanie Z24.28 */
public class Z24_28a {
    public static void main(String[] args) {
        PolarComplex a = new PolarComplex(new Complex(4));
        System.out.println("Pierwiastki kwadratowe z liczby 4:");
        System.out.println(Arrays.toString(a.sqrt()));
        a = new PolarComplex(new Complex(-4));
        System.out.println("Pierwiastki kwadratowe z liczby 4:");
        System.out.println(Arrays.toString(a.sqrt()));
        a = new PolarComplex(new Complex(0, 8));
        System.out.println("Pierwiastki sześcienne z liczby 8i:");
        System.out.println(Arrays.toString(a.cbrt()));
    }
}
```

Zadanie 24.29. Z24_29a.java, Z24_29b.java

a) Jeśli $a \neq 0$, równanie $az^2 + c = 0$ jest równoważne równaniu $z^2 = -\frac{c}{a}$.

Równanie to ma dwa pierwiastki zespolone $z = \sqrt{-\frac{c}{a}}$ (rzeczywiste $z = \pm\sqrt{-\frac{c}{a}}$, gdy $-\frac{c}{a} > 0$) lub pierwiastek dwukrotny $z = 0$ dla $c = 0$.

Do wyznaczenia pierwiastków najwygodniej będzie użyć metody `sqrt()` z jednym parametrem (zob. rozwiązanie zadania 24.26).

```
import java.util.Scanner;
/** Zadanie Z24.29a */
public class Z24_29a {
    public static void main(String[] args) {
        System.out.println("Równanie az^2+c = 0 o współczynnikach
            rzeczywistych");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        System.out.print("c = ");
        double c = input.nextDouble();
        if (a != 0) {
            PolarComplex x = new Complex(-c/a).toPolarComplex();
            x.sqrt(0).toComplex().printlnf("z1 = ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            x.sqrt(1).toComplex().printlnf("z2 = ");
        } else
            System.out.println("Równanie sprzeczne lub nieoznaczone.");
    }
}
```

b) Równanie $z^3 + a = 0$ jest równoważne równaniu $z^3 = -a$. Równanie to ma trzy pierwiastki zespolone $z = \sqrt[3]{-a}$ (w tym jeden jest zawsze rzeczywisty) lub pierwiastek trzykrotny $z = 0$ dla $a = 0$. Pierwiastki obliczymy, stosując metodę `cbrt()` z jednym parametrem (zob. rozwiązanie zadania 24.27).

```
import java.util.Scanner;
/** Zadanie Z24.29b */
public class Z24_29b {
    public static void main(String[] args) {
        System.out.println("Równanie z^3+a = 0 o współczynnikach
            rzeczywistych");
        Scanner input = new Scanner(System.in);
        System.out.print("a = ");
        double a = input.nextDouble();
        PolarComplex x = new Complex(-a).toPolarComplex();
        x.cbrt(0).toComplex().printlnf("z1 = ");
        x.cbrt(1).toComplex().printlnf("z2 = ");
        x.cbrt(2).toComplex().printlnf("z3 = ");
    }
}
```

Zadanie 24.30. Z24_30a.java, Z2430b.java

a) Rozwiązując równanie $az^2 + c = 0$ o współczynnikach zespolonych, postępujemy podobnie jak w rozwiązaniu zadania 24_29a. Różnica polega na innym sposobie wprowadzania danych, wykonywaniu porównań i działań (w sposób specyficzny dla liczb zespolonych). Oto podstawowe różnice:

Współczynniki rzeczywiste	Współczynniki zespolone

Wczytywanie danych
`System.out.print("a = ");`
`double a = input.nextDouble();`
	Wczytywanie danych
	`System.out.print("a.re = ");`
	`double re = input.nextDouble();`
	`System.out.print("a.im = ");`
	`double im = input.nextDouble();`
	`Complex a = new Complex(re, im);`

Porównywanie `a` z zerem `if(a != 0)...`	Porównywanie `a` z zerem `if(!a.equals(Complex.ZERO))...`
Obliczenie prawej strony równania (`-c/a`) (i zamiana na liczbę zespoloną) `PolarComplex x = new Complex (-c/a).toPolarComplex();`	Obliczenie prawej strony równania (`-c/a`) `PolarComplex x = c.opp().div(a).toPolarComplex();`

Obliczenie i wyświetlenie pierwiastków zespolonych
`x.sqrt(0).toComplex().printlnf("z1 = ");`
`x.sqrt(1).toComplex().printlnf("z2 = ");`	

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Scanner;
/** Zadanie Z24.30a */
public class Z24_30a {
    public static void main(String[] args) {
        System.out.println("Równanie az^2+c = 0 o współczynnikach
            zespolonych");
        Scanner input = new Scanner(System.in);
        System.out.print("a.re = ");
        double re = input.nextDouble();
        System.out.print("a.im = ");
        double im = input.nextDouble();
        Complex a = new Complex(re, im);
        a.println("a = ");
        System.out.print("c.re = ");
        re = input.nextDouble();
        System.out.print("c.im = ");
        im = input.nextDouble();
        Complex c = new Complex(re, im);
        c.println("c = ");
        if (!a.equals(Complex.ZERO)) {
            PolarComplex x = c.opp().div(a).toPolarComplex();
            x.sqrt(0).toComplex().println("z1 = ");
            x.sqrt(1).toComplex().println("z2 = ");
        } else
            System.out.println("Równanie sprzeczne lub nieoznaczone.");
    }
}
```

b) Równanie $z^3 + a = 0$ jest równoważne równaniu $z^3 = -a$ (`a.opp()`), a to równanie ma trzy (różne) pierwiastki zespolone $z = \sqrt[3]{-a}$ lub pierwiastek trzykrotny $z = 0 + 0i$ dla $a = 0 + 0i$ (porównaj z rozwiązaniem zadania 24.29b).

```
import java.util.Scanner;
/** Zadanie Z24.30b */
public class Z24_30b {
    public static void main(String[] args) {
        System.out.println("Równanie z^3+a = 0 o współczynnikach
            zespolonych");
        Scanner input = new Scanner(System.in);
        System.out.print("a.re = ");
        double re = input.nextDouble();
        System.out.print("a.im = ");
        double im = input.nextDouble();
        Complex a = new Complex(re, im);
        a.println("a = ");
        PolarComplex x = a.opp().toPolarComplex();
        x.cbrt(0).toComplex().println("z1 = ");
        x.cbrt(1).toComplex().println("z2 = ");
        x.cbrt(2).toComplex().println("z3 = ");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 24.31. Z24_31.java, PolarComplex.java

Metoda `root()` jest realizacją wzoru de Moivre'a:

$$\sqrt[n]{z} = \sqrt[n]{r}\left(\cos\left(\frac{\varphi + 2k\pi}{n}\right) + i\sin\left(\frac{\varphi + 2k\pi}{n}\right)\right), \text{ k = 0, 1, 2… n-1.}$$

```
public PolarComplex root(int n, int k) {
    k %= n;
    return new PolarComplex(Math.pow(r, 1.0/n), (fi+2*k*Math.PI)/n);
}
```

Testując działanie metody, wyznaczymy pierwiastki zespolone piątego stopnia z 32, wśród nich jest jedna wartość rzeczywista (liczba 2).

```
import java.util.Scanner;
/** Zadanie Z24.31 */
public class Z24_31 {
    public static void main(String[] args) {
        System.out.println("Pierwiastki zespolone 5 stopnia z 32");
        PolarComplex a = new Complex(32).toPolarComplex();
        a.toComplex().printlnf("a = ");
        for(int i = 0; i < 5; ++i) {
            PolarComplex x = a.root(5, i);
            System.out.print("x"+i+" = ");
            x.toComplex().printf();
            System.out.println(" = "+x);
        }
    }
}
```

Zadanie 24.32. Z24_32.java, PolarComplex.java

Na podstawie rozwiązania zadania 24.31 budujemy tablicę obiektów `Complex` zawierającą *n* pierwiastków *n*-tego stopnia z liczby zespolonej reprezentowanej przez obiekt `PolarComplex` wywołujący metodę.

```
public Complex[] root(int n) {
    Complex[] tmp = new Complex[n];
    for(int k = 0; k < n; ++k) {
    tmp[k] = new PolarComplex(Math.pow(r, 1.0/n), (fi+2*k*Math.PI)/n).
        toComplex();
    }
    return tmp;
}
```

Testując działanie metody, wyznaczymy pierwiastki ósmego stopnia z *1*, w tym dwie liczby rzeczywiste *1* i *–1*.

```
/** Zadanie Z24.32 */
public class Z24_32 {
    public static void main(String[] args) {
        System.out.println("Pierwiastki zespolone 8 stopnia z 1");
        PolarComplex a = new Complex(1).toPolarComplex();
        a.toComplex().printlnf("a = ");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        for(Complex x: a.root(8)) {
            System.out.print("x = ");
            x.printf();
            System.out.println(" = "+x.toPolarComplex());
        }
    }
}
```

Zadanie 24.33. Z24_33.java

a) Rozwiązanie równania $z^n + 1 = 0$ w zbiorze liczb zespolonych, równoważnego równaniu $z^n = -1$, sprowadza się do obliczenia pierwiastków zespolonych *n*-tego stopnia z liczby *–1*. Liczbę rzeczywistą *–1* zamieniamy na liczbę zespoloną *–1+0i*, a następnie na jej postać biegunową `PolarComplex a = new Complex(-1).toPolarComplex();`. Wywołanie metody `a.root(n)` zwraca nam *n*-elementową tablicę obiektów klasy `Complex`, pierwiastków zespolonych obiektu `a` (odpowiadającego liczbie *–1*) — rozwiązanie naszego równania.

```
import java.util.Scanner;
/** Zadanie Z24.33 */
public class Z24_33a {
    public static void main(String[] args) {
        System.out.println("Pierwiastki równania z^n+1 = 0");
        Scanner input = new Scanner(System.in);
        System.out.print("n = ");
        int n = input.nextInt();
        PolarComplex a = new Complex(-1).toPolarComplex();
        a.toComplex().printlnf("a = ");
        int i = 1;
        for(Complex x: a.root(n)) {
            System.out.print("z"+i+" = ");
            ++i;
            x.printf();
            System.out.println(" = "+x.toPolarComplex());
        }
    }
}
```

Przedstawione rozwiązanie możemy uogólnić dla wszystkich równań o postaci $z^n + a = 0$.

b) W rozwiązaniu zadania 24.33a zmieniamy dwa wiersze programu na następujące:

```
System.out.println("Pierwiastki równania z^n+i = 0");
```

oraz

```
PolarComplex a = new Complex(0, -1).toPolarComplex();
```

c) Rozwiązujemy w zbiorze liczb zespolonych równanie kwadratowe $x^2 + ix + 1 = 0$ (zob. rozwiązanie zadania 24.17) i otrzymujemy dwa pierwiastki zespolone x_1 i x_2.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/* Równanie kwadratowe ax^2+bx+c = 0, a = 1, b = i, c = 1 */
Complex a = new Complex(1);
Complex b = new Complex(0, 1);
Complex c = new Complex(1);
Complex delta = b.sqr().sub(new Complex(4).mult(a).mult(c));
Complex x1 = b.opp().sub(delta.sqrt()).div(new Complex(2).mult(a));
Complex x2 = b.opp().add(delta.sqrt()).div(new Complex(2).mult(a));
```

Następnie rozwiązujemy dwa równania: $z^n = x_1$ i $z^n = x_2$, wzorując się na rozwiązaniu zadania 24.33a (24.33b). Najpierw obliczymy serię n pierwiastków pierwszego równania ($z^n = x_1$).

```
int i = 1;
/* Równanie z^n = x1 */
PolarComplex z = x1.toPolarComplex();
for(Complex x: z.root(n)) {
    System.out.print("z"+i+" = ");
    ++i;
    x.printf();
    System.out.println(" = "+x.toPolarComplex());
}
```

Kolejne pierwiastki wyznaczymy w ten sam sposób, na podstawie drugiego równania ($z^n = x_2$).

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozdział 8.

Rozwiązania zadań

25. Operacje na plikach tekstowych

Zadanie 25.1. Z25_1.java

Z pakietu `java.io` importujemy dwie klasy `IOException` i `FileWriter`.

W nagłówku metody `main()` umieszczamy słowo kluczowe `throws` i nazwę klasy `IOException` — użyte metody mogą zgłosić wyjątki tej klasy, a metoda `main()` nie będzie ich przechwytywać i obsługiwać. Jeśli wyjątek zostanie zgłoszony, to działanie programu zostanie przerwane i maszyna wirtualna Javy wyświetli stosowny komunikat (np. mówiący o próbie użycia nazwy pliku *?.txt*):

```
Exception in thread "main" java.io.FileNotFoundException: ?.txt (Nazwa pliku, nazwa
    katalogu lub składnia etykiety woluminu jest niepoprawna)
      at java.io.FileOutputStream.open(Native Method)
      at java.io.FileOutputStream.<init>(FileOutputStream.java:194)
      at java.io.FileOutputStream.<init>(FileOutputStream.java:84)
      at java.io.FileWriter.<init>(FileWriter.java:46)
      at Z25_1.main(Z25_1.java:6)
```

Tworzymy obiekt `fout` klasy `FileWriter` skojarzony z plikiem *tekst.txt* w bieżącym folderze. Jeśli plik wcześniej nie istniał, to zostanie utworzony, natomiast w przypadku pliku istniejącego jego dotychczasowa zawartość zostanie usunięta.

Metoda `write()` wpisuje do pliku łańcuch znaków. Na koniec musimy pamiętać o zamknięciu pliku (metoda `close()`).

```
import java.io.IOException;
import java.io.FileWriter;
/** Zadanie Z25.1 */
public class Z25_1 {
    public static void main(String[] args) throws IOException {
        FileWriter fout = new FileWriter("tekst.txt");
        fout.write("Programowanie obiektowe");
        fout.close();
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Wszelkie sytuacje błędne, związane z otwarciem pliku lub zapisem do pliku, wygenerują wyjątek, który przerwie pracę programu. Wyjątki możemy przechwycić i obsłużyć.

```
public static void main(String[] args) {
    try {
        FileWriter fout = new FileWriter("tekst.txt");
        fout.write("Programowanie obiektowe");
        fout.close();
    } catch (IOException e) {
       System.err.println("Błąd "+e);
    }
}
```

Zadanie 25.2. Z25_2.java

Uwagi przekazane w rozwiązaniu zadania 25.1 pozostają aktualne; istotną różnicą jest sposób tworzenia obiektu `FileWriter fout = new FileWriter("tekst.txt", true);` — wartość `true` drugiego parametru w konstruktorze oznacza otwarcie pliku w trybie dopisywania na końcu pliku (podanie wartości `false` spowoduje, że dotychczasowa zawartość pliku zostanie usunięta — jest to wartość przyjmowana domyślnie, gdy pominiemy drugi parametr).

Koniec wiersza uzyskamy, wpisując do strumienia łańcuch zawierający znaki *CR+LF* `fout.write("\r\n")` lub każdy znak osobno: `fout.write('\r')` (*CR*, kod ASCII *13*), `fout.write('\n')` (*LF*, kod ASCII *10*).

```
/** Zadanie Z25.2 */
import java.io.IOException;
import java.io.FileWriter;
public class Z25_2 {
    public static void main(String[] args) {
       try {
           FileWriter fout = new FileWriter("tekst.txt", true);
           fout.write(" w języku Java\r\n");
           fout.write("jest bardzo interesujące.");
           fout.close();
       } catch (IOException e) {
           System.err.println("Błąd "+e);
       }
    }
}
```


Uwaga

Po jednokrotnym wykonaniu programu z zadania 25.1 i 25.2 plik tekstowy *tekst.txt* powinien zawierać następujący tekst:

```
Programowanie obiektowe w języku Java
jest bardzo interesujące.
```

Do tego pliku odwołamy się w dalszych zadaniach.

Zadanie 25.3. Z25_3.java

Podobnie jak w rozwiązaniu zadania 25.1, otwieramy plik tekstowy *silnia.txt* do zapisywania. W pętli obliczamy wartości silni i zapisujemy wyniki do pliku. Argumentem

Ebookpoint.pl kopia dla: pawher@o2.pl

metody `write()` może być pojedynczy znak (liczba typu `int` traktowana jako kod znaku), tablica znaków lub łańcuch znaków (`String`). Z dwóch liczb i tekstu, wykorzystując właściwości operatora konkatenacji tekstów, budujemy wiersz odpowiedzi i zapisujemy go do pliku `fout.write(i+"! = "+silnia)`. Koniec linii wpisujemy warunkowo we wszystkich wierszach odpowiedzi z wyjątkiem ostatniego (dwunastego wiersza) `if (i < 12) fout.write("\r\n")`. Na koniec zamykamy plik (`fout.close()`).

```
/** Zadanie Z25.3 */
import java.io.IOException;
import java.io.FileWriter;
public class Z25_3 {
    public static void main(String[] args) {
       try {
           FileWriter fout = new FileWriter("silnia.txt");
           int silnia = 1;
           for(int i = 1; i < 13; ++i) {
               silnia *= i;
               fout.write(i+"! = "+silnia);
               if (i < 12)
                   fout.write("\r\n");
           }
           fout.close();
       } catch (IOException e) {
           System.err.println("Błąd "+e);
       }
    }
}
```

Zadanie 25.4. Z25_4.java

Zadanie to rozwiązujemy podobnie jak zadanie 25.3. Liczby zamieniamy na sformatowane łańcuchy znaków przy użyciu metody `format()` z klasy `String`:

- `String.format("%2d\t", n)` — liczba całkowita `n` zostanie umieszczona w dwóch pierwszych znakach łańcucha, trzecim znakiem będzie znak tabulatora `'\t'`.
- `String.format("%.12f\t", Math.sqrt(n))` — liczba zmiennoprzecinkowa, pierwiastek kwadratowy z `n`, zostanie umieszczona na początku łańcucha z dwunastoma miejscami po przecinku; za liczbą będzie dodany znak tabulatora `'\t'`.
- `String.format("%.12f\t", Math.cbrt(n))` — liczba zmiennoprzecinkowa, pierwiastek sześcienny z `n`, zostanie umieszczona w łańcuchu z dwunastoma miejscami po przecinku.

Znak końca linii (`fout.write("\r\n")`) wpisywany jest warunkowo we wszystkich wierszach z wyjątkiem wiersza ostatniego.

```
/** Zadanie Z25.4 */
import java.io.IOException;
import java.io.FileWriter;
public class Z25_4 {
    public static void main(String[] args) {
        try {
            FileWriter fout = new FileWriter("pierwiastki.txt");
            for(int n = 2; n <= 15; ++n) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                fout.write(String.format("%2d\t", n));
                fout.write(String.format("%.12f\t", Math.sqrt(n)));
                fout.write(String.format("%.12f", Math.cbrt(n)));
                if (n < 15)
                    fout.write("\r\n");
            }
            fout.close();
        } catch (IOException e) {
            System.err.println("Błąd "+e);
        }
    }
}
```

Zadanie 25.5. Z25_5.java

Utworzenie pliku możemy zrealizować podobnie jak w zadaniu 25.3 (lub 25.4). Liczby losujemy w pętli powtarzającej się sto razy, korzystając z metody `random()` z klasy `Math`. Wylosowaną liczbę `n` zamieniamy na łańcuch znaków, wykonując dodawanie (konkatenację) odstępu do liczby `n+" "`. Wynik tego dodawania zapisujemy w pliku `fout.write(n+" ")` — ten odstęp będzie oddzielał kolejne liczby i pojawi się na końcu pliku, za ostatnią liczbą. Możemy sytuację odwrócić (`fout.write(" "+n)`) i wtedy dodatkowy odstęp pojawi się na początku pliku, przed pierwszą liczbą. Ta sytuacja może być korzystniejsza podczas odczytywania liczb z pliku.

```
/** Zadanie Z25.5 */
import java.io.IOException;
import java.io.FileWriter;
public class Z25_5 {
    public static void main(String[] args) {
        try {
            FileWriter fout = new FileWriter("100.txt");
            for(int i = 0; i <= 100; ++i) {
                int n = (int)(1+100*Math.random());
                fout.write(n+" ");
            }
            fout.close();
        } catch (IOException e) {
            System.err.println("Błąd "+e);
        }
    }
}
```

Zadanie to możemy również rozwiązać, stosując obiekty i metody klasy `PrintWriter`.

```
import java.io.IOException;
import java.io.PrintWriter;
public class Z25_5a {
    public static void main(String[] args) {
        try {
           PrintWriter fout = new PrintWriter("100.txt");
           for(int i = 0; i <= 100; ++i) {
               int n = (int)(1+100*Math.random());
               fout.print(" "+n);
           }
           fout.close();
        } catch (IOException e) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            System.err.println("Błąd "+e);
        }
    }
}
```


Uwaga

W klasie `PrintWriter` mamy do dyspozycji metody `print()`, `println()`, `printf()` i `format()`, działające w taki sam sposób jak wymieniane już metody obiektu `out` klasy `System`. Dostępna jest również metoda `write()` znana z klasy `FileWriter`. Ułatwia to zapisywanie różnych danych do plików tekstowych.

Zadanie 25.6. Z25_6.java

Plik zapiszemy, używając metod z klasy `PrintWriter` (zob. rozwiązanie zadania 25.5). Zmienna `x = 10*Math.random()` przyjmuje wartości z przedziału $\langle 0, 10)$. Losowanie odbywa się w pętli (typu `do {...} while (x < 0.01)`), więc dla wartości mniejszych od *0,01* losowanie jest powtarzane. Zmienna `n = (int)(2+7*Math.random())` przyjmuje wartości całkowite od *2* do *8*.

Metoda `printf()` formatuje parę liczb i zapisuje je do pliku (`fout.printf("%.2f %d", x, n)`). Przejście do nowego wiersza jest realizowane przy użyciu metody `println()`, pod warunkiem że nie jest to ostatni wiersz pliku.

```
/** Zadanie Z25.6 */
import java.io.IOException;
import java.io.PrintWriter;
public class Z25_6 {
    public static void main(String[] args)  {
        try {
           PrintWriter fout = new PrintWriter("dane.txt");
           for(int i = 0; i < 50; ++i) {
               double x;
               int n;
               do {
                   x = 10*Math.random();
               } while (x < 0.01);
               n = (int)(2+7*Math.random());
               fout.printf("%.2f %d", x, n);
               if (i < 49)
                   fout.println();
           }
           fout.close();
        } catch (IOException e) {
           System.err.println("Błąd "+e);
        }
    }
}
```

Zadanie 25.7. Z25_7.java

Z pakietu `java.io` importujemy dwie klasy: `IOException` i `FileWriter`. Tworzymy obiekt `fin` klasy `FileReader` skojarzony z plikiem, np. *silnia.txt*, w bieżącym folderze. Jeśli plik nie istnieje, to zostanie zgłoszony wyjątek `FileNotFoundException`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Tworzymy zmienną typu `int` do przechowywania kodu odczytanego znaku (`int i`). W pętli odczytujemy kod znaku `i = fin.read()` i jeśli znak został odczytany (`i` nie jest równe `-1`), to wyświetlamy znak w konsoli `System.out.print((char)i)`. Niezbędne jest rzutowanie kodu znaku na typ `char`. Odczytana wartość zmiennej `i` równa `-1` oznacza koniec pliku (nie ma dalszych znaków do odczytywania). Na koniec należy zamknąć plik (`fin.close()`).

```
/** Zadanie Z25.7 */
import java.io.IOException;
import java.io.FileReader;
public class Z25_7 {
    public static void main(String[] args) {
        try {
            FileReader fin = new FileReader("silnia.txt");
            int i;
            do {
               i = fin.read();
               if (i != -1) System.out.print((char)i);
            } while (i != -1);
            fin.close();
        } catch (IOException e) {
            System.err.println("Błąd "+e);
        }
    }
}
```

Korzystając z metody `ready()`, odczytywanie i wyświetlanie zawartości pliku tekstowego możemy zrealizować w następujący sposób:

```
FileReader fin = new FileReader("silnia.txt");
while (fin.ready())
    System.out.print((char)fin.read());
fin.close();
```

Zadanie 25.8. Z25_8.java

Wywołując metodę `readLine()`, przekazujemy jako parametr `f` obiekt klasy `FileReader` reprezentujący otwarty plik tekstowy. Metoda `readLine()` odczytuje z pliku kolejne znaki, aż do chwili, gdy zostanie osiągnięty *koniec linii* (*CR+LF*, zob. rozwiązanie zadania 25.2) lub *koniec pliku* (metoda `read()` zwróci `-1`). Odczytywanie znaków odbywa się w nieskończonej pętli. Z odczytywanych znaków budowany jest łańcuch znaków, który będzie zwrócony jako odczytany wiersz pliku tekstowego.

```
private static String readLine(FileReader f) throws IOException {
    StringBuilder s = new StringBuilder();
    int c;
    while (true){
        c = f.read();//System.out.println(c);
        if (c == 13) continue;
        if (c == 10 || c == -1)
            return s.toString();
        s.append((char) c);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Przeanalizujmy działanie pętli. Gdy następuje wywołanie metody `read()`, zmienna `c` przyjmuje wartość kodu odczytanego znaku lub `-1`. Jeśli kod ma wartość *13* (*CR*), to przechodzimy do kolejnego cyklu (instrukcja `continue`), spodziewając się znaku (*LF*, kod *10*). Jeśli kod ma wartość *10* (*LF*), to przerywamy działanie metody, zwracając łańcuch znaków (`return s.toString()`) — odczytany wiersz z pliku. Podobnie postąpimy, gdy osiągniemy koniec pliku (`c == -1`). W pozostałych przypadkach dodajemy do łańcucha `s` odczytany znak `s.append((char) c)`.

Obsługa wyjątków wygenerowanych podczas działania metody `readLine()` zostanie przekazana do metody wywołującej (w tym przypadku do metody `main()`).

Analiza kodu reszty programu nie wymaga szerszego komentarza — podobna sytuacja została omówiona w rozwiązaniu zadania 25.7.

```
/** Zadanie Z25.8 */
import java.io.IOException;
import java.io.FileReader;
public class Z25_8 {

    /* Tu wstaw kod metody readLine(). */

    public static void main(String[] args) {
        try {
            FileReader fin = new FileReader("silnia.txt");
            while (fin.ready())
                System.out.println(readLine(fin));
            fin.close();
        } catch (IOException e) {
            System.err.println("Błąd "+e);
        }
    }
}
```

Zadanie 25.9. Z25_9.java

W klasie `BufferdReader` zdefiniowano metodę `readLine()`, którą wykorzystamy do odczytywania wiersza tekstu (z konsoli i z pliku).

Tworzymy obiekt `isr` klasy `InputStreamReader` będący połączeniem bajtowego strumienia danych (np. `System.in` — standardowe wejście) ze strumieniem znakowym:

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

Następnie tworzymy obiekt `br` klasy `BuffeeredReader` opakowujący obiekt `isr`:

```
BufferedReader br = new BufferedReader(isr);
```

Możemy ten zapis ująć w jednym wierszu programu (bez tworzenia zmiennej obiektowej `isr`):

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Teraz możemy odczytywać wiersze tekstu z konsoli — w ten sposób wprowadzimy nazwę pliku `String filename = br.readLine()`. Zamykamy utworzone strumienie

Ebookpoint.pl kopia dla: pawher@o2.pl

(metodą `close()`) i tworzymy obiekt `File file = new File(filename+".java")`. Metoda `exists()` z klasy `File` umożliwi nam sprawdzenie, czy plik o podanej nazwie istnieje.

Jeśli plik istnieje, to możemy przystąpić do odczytywania go wierszami i wyświetlania tych wierszy w konsoli wraz z ich numerami.

```
FileReader fr = new FileReader(file);
br = new BufferedReader(fr);
String s;
int n = 0; // licznik wierszy
while((s = br.readLine()) != null) {
    System.out.printf("%3d. %s\n", ++n, s);
}
fr.close();
```

Zwróćmy uwagę na sposób utworzenia obiektu `fr` — w konstruktorze parametrem jest obiekt `file` (dotychczas, w zadaniach 25.7 i 25.8, parametrem tym była nazwa pliku). Obiekt `br` klasy `BufferedReader` opakowuje obiekt `fr` (klasy `FileReader`) i umożliwia odczytywanie wierszy z pliku.

Do wyświetlania numeru wiersza (`n`) i zawartości wiersza (`s`) użyjemy metody `printf()` umożliwiającej sformatowanie wyniku: `System.out.printf("%3d. %s\n", ++n, s)`.

```
/** Zadanie Z25.9 */
import java.io.InputStreamReader;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
class Z25_9 {
    public static void main(String args[]) throws Exception {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        System.out.print("Podaj nazwę pliku bez rozszerzenia: ");
        String filename = br.readLine();
        br.close();
        isr.close();
        File file = new File(filename+".java");
        if (file.exists()) {
            FileReader fr = new FileReader(file);
            br = new BufferedReader(fr);
            String s;
            int n = 0;
            while((s = br.readLine()) != null)
                System.out.printf("%3d. %s\r\n", ++n, s);
            fr.close();
        } else
            System.out.println("Nie znaleziono pliku: "+filename
                +".java");
    }
}
```


Uwaga

Do wczytywania danych z konsoli i wierszy z pliku możemy użyć obiektów klasy `Scanner`. Przedstawimy to w rozwiązaniu kolejnego zadania (25.10).

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 25.10. Z25_10.java

Do odczytywania danych z konsoli użyjemy obiektu (skanera) klasy `Scanner` skojarzonego ze strumieniem `System.in` (`Scanner cin = new Scanner(System.in)`) i metody `nextLine()`. Z konsoli odczytamy nazwę pliku: `String filename = cin.nextLine()`.

Po sprawdzeniu, czy plik istnieje, tworzymy plik wyjściowy o tej samej nazwie i rozszerzeniu *.txt* oraz obiekt `fout` klasy `PrintWriter`, umożliwiający zapisywanie danych do tego pliku:

```
PrintWriter fout = new PrintWriter(new File(filename+".txt"));
```

Tworzymy skaner skojarzony z plikiem danych (`Scanner fin = new Scanner(file)`), służący do odczytywania danych z tego pliku.

Odczytywanie wierszy z pliku zorganizujemy w pętli (`while`) ze sprawdzaniem warunku na początku — metoda `hasNextLine()` zwraca wartość `true`, gdy w pliku jest kolejna linia do odczytania. Odczytany z pliku wiersz (`s = fin.nextLine()`) wraz z poprzedzającym go numerem (`fout.printf("%3d. %s\r\n", ++n, s)`) zapisujemy w pliku wyjściowym.

```
/** Zadanie Z25.10 */
import java.util.Scanner;
import java.io.File;
import java.io.PrintWriter;
class Z25_10 {
    public static void main(String args[]) throws Exception {
        Scanner cin = new Scanner(System.in);
        System.out.print("Podaj nazwę pliku bez rozszerzenia: ");
        String filename = cin.nextLine();
        cin.close();
        File file = new File(filename+".java");
        if (file.exists()) {
            PrintWriter fout
                = new PrintWriter(new File(filename+".txt"));
            Scanner fin = new Scanner(file);
            String s;
            int n = 0;
            while(fin.hasNextLine()) {
                s = fin.nextLine();
                System.out.printf("%3d. %s\r\n", ++n, s);
                fout.printf("%3d. %s\r\n", ++n, s);
            }
            fin.close();
            fout.close();
        } else
            System.out.println("Nie znaleziono pliku: "+filename+".java");
    }
}
```

Zadanie 25.11. Z25_11.java

Do odczytywania danych z pliku użyjemy obiektu i metod z klasy `Scanner` (zob. rozwiązanie zadania 25.10). Metoda `hasNextInt()` sprawdza, czy następnym tokenem w strumieniu jest liczba całkowita, a metoda `nextInt()` odczytuje liczbę całkowitą ze

Ebookpoint.pl kopia dla: pawher@o2.pl

strumienia. Jeśli w strumieniu znajdzie się token różny od liczby całkowitej, to wczytywanie i sumowanie danych zostanie zakończone. W przypadku poprawnie zbudowanego pliku zakończenie sumowania nastąpi po odczytaniu z niego wszystkich liczb.

```
/** Zadanie Z25.11 */
import java.util.Scanner;
import java.io.File;
class Z25_11 {
    public static void main(String args[]) throws Exception {
        File file = new File("sto.txt");
        Scanner fin = new Scanner(file);
        int suma = 0, n;
        while(fin.hasNextInt()) {
            n = fin.nextInt();
            suma += n;
            System.out.printf("%d ", n);
        }
        fin.close();
        System.out.println();
        System.out.println("Suma liczb: "+suma);
    }
}
```

Nieznaczna modyfikacja kodu pozwoli na pomijanie tych tokenów, które nie są liczbami całkowitymi:

```
while(fin.hasNext()) {
    if (fin.hasNextInt()) {
        n = fin.nextInt();
        suma += n;
        System.out.printf("%d ", n);
    } else
        fin.next();
}
```

Po tej zmianie możemy sprawdzić działanie programu dla plików *silnia.txt* (zsumowane zostaną tylko wartości silni, tokeny o postaci *5!* lub = będą pominięte), *pierwiastek.txt* (suma liczb z pierwszej kolumny), *tekst.txt* (otrzymamy sumę równą zeru, bo ten plik nie zawiera żadnej liczby) itp.

Zadanie 25.12. Z25_12.java

Zakładamy, że plik wejściowy jest zbudowany poprawnie. Wczytywanie danych zrealizujemy tak jak w rozwiązaniu zadania 25.11. Wprowadzimy trzy zmienne całkowite: `min` — do przechowywania wartości minimalnej, `cmin` — licznik wystąpień wartości minimalnej, `n` — aktualnie odczytywana liczba.

Pierwszą odczytaną liczbę uznajemy za minimalną, licznik wartości minimalnych ustawiamy na *1*. Następnie odczytujemy kolejne liczby. Wartości większe od minimalnej pomijamy; gdy napotkamy wartość równą minimalnej, to zwiększamy licznik (`cmin`) o *1*. Jeśli odczytana liczba jest mniejsza od minimalnej, to staje się nową wartością minimalną i w tym przypadku ponownie ustawiamy licznik (`cmin`) na *1*.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zadanie Z25.12 */
import java.util.Scanner;
import java.io.File;
class Z25_12 {
    public static void main(String args[]) throws Exception {
        File file = new File("sto.txt");
        Scanner fin = new Scanner(file);
        int min = 0, cmin = 0;
        if (fin.hasNextInt()) {
            min = fin.nextInt();
            cmin = 1;
        }
        while(fin.hasNextInt()) {
            int n = fin.nextInt();
            if (n > min)
                continue;
            else if (n == min)
                ++cmin;
            else {
                min = n;
                cmin = 1;
            }
        }
        fin.close();
        System.out.println("Wartość minimalna: "+min);
        System.out.println("Liczba wystąpień: "+cmin);
    }
}
```

Instrukcja warunkowa wewnątrz pętli została utworzona zgodnie z wcześniejszym opisem. Możemy ten fragment kodu zrealizować inaczej (bez użycia instrukcji `continue`).

```
if (n == min)
    ++cmin;
else if (n < min) {
    min = n;
    cmin = 1;
}
```

Zadanie 25.13. Z25_13.java

Plik *dane.txt* zawiera pewną liczbę wierszy tekstu, a w każdym wierszu zapisane są dwie liczby oddzielone białym znakiem (np. odstępem). Pierwsza z tych liczb jest liczbą zmiennoprzecinkową, a druga — liczbą całkowitą. Zakładamy, że budowa pliku jest poprawna. Do odczytywania liczb z pliku wykorzystamy obiekt `fin` klasy `Scanner` i metody `nextDouble()` (do odczytania pierwszej liczby w wierszu) oraz `nextInt()` (do odczytania kolejnej liczby w wierszu).

Deklarujemy i inicjujemy zmienną `suma`, służącą do obliczania sumy iloczynów par liczb zapisanych w pliku (`double suma = 0.0`). Dopóki w pliku są kolejne tokeny do odczytania (`fin.hasNext()` zwraca wartość `true`), w pętli typu `while` odczytujemy z pliku liczbę zmiennoprzecinkową (`double x = fin.nextDouble();`) i liczbę całkowitą (`int n = fin.nextInt()`), obliczamy iloczyn odczytanych liczb i dodajemy wynik do sumy (`suma += x*n`). Po odczytaniu wszystkich liczb zamykamy plik (`fin.close()`).

Ebookpoint.pl kopia dla: pawher@o2.pl

Otwieramy ponownie plik *dane.txt*, tym razem w trybie do dopisywania (`FileWriter fout = new FileWriter(file, true)`) i dopisujemy na końcu pliku odpowiedź.

```
/** Zadanie Z25.13 */
import java.util.Scanner;
import java.io.File;
import java.io.FileWriter;
class Z25_13 {
    public static void main(String args[]) throws Exception {
        File file = new File("dane.txt");
        Scanner fin = new Scanner(file);
        double suma = 0.0;
        while(fin.hasNext()) {
            double x = fin.nextDouble();
            int n = fin.nextInt();
            suma += x*n;
        }
        fin.close();
        System.out.printf("Suma iloczynów: %.2f\r\n", suma);
        /* Dopisanie wyniku do pliku */
        FileWriter fout = new FileWriter(file, true);
        fout.write("\r\nSuma iloczynów: ");
        fout.write(String.format("%.2f", suma));
        fout.close();
    }
}
```

26. Tablice jednowymiarowe i wielomiany

Zadanie 26.1. Z26_1.java

Korzystając z obiektu `cin` klasy `Scanner`, wczytujemy stopień wielomianu (`n`) i tworzymy tablicę *n+1*-elementową do przechowywania współczynników wielomianu (`double[] w = new double[n+1]`). W pętli (typu `for` — znamy liczbę współczynników) odczytujemy z konsoli wartości współczynników wielomianu `w[i]`, zaczynając od wyrazu wolnego. Ostatni współczynnik (`w[n]`) powinien być różny od zera (sprawdzenie tego faktu w przykładzie pominięto). W takim przypadku możemy przerwać działanie programu lub zmniejszyć stopień wielomianu i związany z tym rozmiar tablicy, odcinając końcowe wyrazy o wartości *0* (powrócimy do tego w kolejnych zadaniach).

Obliczenia wartości wielomianu zrealizujemy w pętli ze sprawdzaniem warunku na końcu (argument $x = 0$ powoduje przerwanie pętli). Do obliczeń zastosujemy prywatą metodę `horner()` z dwoma parametrami — tablicą współczynników wielomianu `w` i argumentem `x` (typu `double`).

Metoda Hornera obliczania wartości wielomianu polega na następującym spostrzeżeniu:

$$w(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0 = \left(\left(\ldots\left(\left(a_n x + a_{n-1}\right)x + a_{n-2}\right)x + \ldots\right)x + a_1\right)x + a_0 .$$

Ebookpoint.pl kopia dla: pawher@o2.pl

Aby uprościć zapis algorytmu, wprowadzimy małą modyfikację wzoru:

$$w(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0 = ((...(((0x + a_n)x + a_{n-1})x + a_{n-2})x + ...)x + a_1)x + a_0 .$$

Początkowo przyjmujemy `wart = 0`. W pierwszym cyklu obliczamy wyrażenia (`wart *= x; wart += w[i];`) $0x + a_n = a_n$ — zmienna `wart` zawiera wartość współczynnika a_n — w drugim cyklu liczymy $a_n x + a_{n-1}$, w trzecim $(a_n x + a_{n-1})x + a_{n-2}$ itd. Po skończonym cyklu obliczeń zmienna `wart` zawiera obliczoną wartość wielomianu. Zauważmy, że współczynniki odczytujemy od końca tablicy do początku (od współczynnika przy jednomianie najwyższego stopnia do wyrazu wolnego).

```
/** Zadanie Z26.1 */
import java.util.Scanner;
public class Z26_1 {
    private static double horner (double[] w, double x) {
        int st = w.length-1;
        double wart = 0;
        for(int i = st; i >= 0; --i) {
            wart *= x;
            wart += w[i];
        }
        return wart;
    }
    public static void main(String[] args) {
        Scanner cin = new Scanner(System.in);
        System.out.print("Podaj stopień wielomianu, n = ");
        int n = cin.nextInt();
        double[] w = new double[n+1];
        System.out.println("Wprowadź współczynniki wielomianu: ");
        for(int i = 0; i <= n; ++i) {
            System.out.print("w["+i+"] = ");
            w[i] = cin.nextDouble();
        }
        System.out.println("Obliczanie wartości wielomianu w(x):");
        System.out.println("(argument x=0 - zakończenie obliczeń)");
        double x;
        do {
            System.out.print("x = ");
            x = cin.nextDouble();
            System.out.printf("w(%.2f) = %.4f\r\n", x, horner(w,x));
        } while (x != 0);
    }
}
```

W kolejnych zadaniach obliczenia wykonamy dla ustalonych wielomianów. Czytelnik może zmodyfikować kod programów tak, aby współczynniki wielomianów można było wprowadzać z konsoli (zob. rozwiązanie zadania 26.1).

W przykładach podajemy współczynniki całkowite, ale dopuszczalne jest również użycie współczynników ułamkowych.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 26.2. Z26_2.java

Do obliczania wartości wielomianu zastosujemy metodę `horner()` omówioną w rozwiązaniu zadania 26.1. Dane i wyniki zapiszemy w pliku, używając metod z klasy `PrintWriter`. Komentarze w kodzie źródłowym objaśniają kolejne etapy rozwiązywania zadania.

```
/** Zadanie Z26.2 */
import java.io.IOException;
import java.io.PrintWriter;
public class Z26_2 {

    /* Tu wstaw kod metody horner z zadania 26.1. */

    public static void main(String[] args) throws IOException {
        /* Wielomian 2x^3+5x^2-x+3 */
        double[] w = {3, -1, 5, 2};
        /* Przedział i krok obliczeń */
        double a = -2.0;
        double b = 3.0;
        double h = 0.125;
        /* Zapisywanie do pliku */
        PrintWriter fout = new PrintWriter("wielomian.txt");
        /* Stopień wielomianu */
        fout.printf("%d", w.length-1);
        fout.println();
        /* Współczynniki wielomianu */
        for(int i = 0; i < w.length; ++i)
            fout.printf("%.2f ", w[i]);
        fout.println();
        /* Przedział i krok obliczeń */
        fout.printf("%.2f %.2f %.3f", a, b, h);
        fout.println();
        /* Zapisywanie wartości wielomianu */
        double x = a;
        do {
            fout.printf("%.3f %.4f", x, horner(w,x));
            fout.println();
            x += h;
        } while (x < b);
        /* Zamknięcie pliku */
        fout.close();
    }
}
```

Zadanie 26.3. Z26_3.java

Jeśli dodajemy dwa wielomiany stopni *m* i *n*, to otrzymujemy wielomian co najwyżej stopnia *max(m, n)*, np.

$$\left(3x^2 - 2x + 4\right) + \left(x^3 + x - 5\right) = x^3 + 3x^2 - x - 1\,.$$

W przypadku $m = n$ suma wielomianów może mieć niższy stopień (gdy współczynniki przy x w najwyższej potędze są liczbami przeciwnymi), np.

$$\left(3x^2 - 2x + 4\right) + \left(-3x^2 + x - 5\right) = 0x^2 - x - 1 = -x - 1\,.$$

Ebookpoint.pl kopia dla: pawher@o2.pl

Współczynniki wielomianów możemy wprowadzić z klawiatury, tak jak w rozwiązaniu zadania 26.1. W przykładzie ograniczymy się do dwóch wielomianów, których współczynniki umieścimy bezpośrednio w kodzie programu (tablice `a` i `b`). Podczas wypisywania wielomianu w konsoli (a właściwie jego współczynników) wykorzystamy statyczną metodę `toString()` z klasy `Arrays`, która zamienia tablicę na łańcuch znaków zawierający elementy tablicy oddzielone przecinkami, umieszczone w nawiasie prostokątnym.

Tworzymy tablicę `s` do przechowywania wyniku (sumy), o wymiarze takim, jaki ma tablica przechowująca wielomian wyższego stopnia (`max(a.length, b.length)`). Z obu tablic bierzemy kolejno elementy o indeksach od `0` do `min(a.length, b.length)`, dodajemy je i sumę wpisujemy w odpowiednim miejscu tablicy przechowującej wynik. Następnie z tej tablicy, która ma większy rozmiar, przepisujemy resztę elementów do tablicy `s`.

Jeśli ostatni wyraz tablicy `s` ma wartość 0, co może zdarzyć się wyłącznie wtedy, gdy długości tablic `a` i `b` (stopnie dodawanych wielomianów) są równe, usuwamy ten wyraz z tablicy (postępowanie będzie prowadzone do chwili uzyskania ostatniego współczynnika w tablicy różnego od zera lub uzyskania wielomianu stopnia zerowego):

```
if(s[s.length-1] == 0) {
    int n = s.length;
    while (s[n-1] == 0 && n > 1) --n;
        s = Arrays.copyOf(s, n);
}
```

Skrócenie tablicy możemy zrealizować, używając statycznej metody `copyOf()` z klasy `Arrays` — pierwszy parametr jest kopiowaną tablicą, a drugi liczbą kopiowanych elementów. Wynik kopiowania przypisujemy do tablicy `s`.

```
/** Zadanie Z26.3 */
import java.util.Arrays;
public class Z26_3 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 3, 5, 2, 1};
        System.out.println("a = "+Arrays.toString(a));
        double[] b = {3, -4, 2, 1};
        System.out.println("b = "+Arrays.toString(b));
        double[] s = new double[Math.max(a.length, b.length)];
        int i = 0;
        for(i = 0; i < Math.min(a.length, b.length); ++i)
            s[i] = a[i]+b[i];
        if (a.length > b.length)
            while (i < a.length) {
                s[i] = a[i];
                ++i;
            }
        else if (a.length < b.length)
            while (i < b.length) {
                s[i] = b[i];
                ++i;
            }
        else if(s[s.length-1] == 0) {
            int n = s.length;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            while (s[n-1] == 0 && n > 1) --n;
            s = Arrays.copyOf(s, n);
        }
        System.out.println("Suma wielomianów: ");
        System.out.println("a+b = "+Arrays.toString(s));
    }
}
```

Możemy zbudować metodę `korekta()`, która usunie z końca tablicy wszystkie elementy o wartości `0` (elementy zerowe wewnątrz tablicy muszą bezwzględnie pozostać).

```
private static double[] korekta(double[] w) {
    int n = w.length;
    while (w[n-1] == 0 && n > 1) --n;
    return Arrays.copyOf(w, n);
}
```

W szczególnym przypadku zwracana tablica może zawierać tylko jeden element odpowiadający wielomianowi zerowego stopnia. Metodę tę możemy wykorzystać do skorygowania ostatecznego wyniku (`if(s[s.length-1] == 0) s = korekta(s);`) lub danych wejściowych (np. `a = korekta(a)`) wprowadzonych przez użytkownika programu.

Zadanie 26.4. Z26_4.java

Zadanie to rozwiążemy podobnie jak zadanie 26.3. Należy zwrócić uwagę na przypadek, gdy odejmowany wielomian ma wyższy stopień niż wielomian, od którego odejmujemy.

```
for(i = 0; i < Math.min(a.length, b.length); ++i)
    s[i] = a[i]-b[i];  // obliczanie różnicy
if (a.length > b.length)
    while (i < a.length) {
        s[i] = a[i];
        ++i;
    }
else if (a.length < b.length)
    while (i < b.length) {
        s[i] = -b[i];  // zmiana znaku współczynników
        ++i;
    }
else if(s[s.length-1] == 0)
    s = korekta(s);
```

Zmiany zaznaczono dodatkowymi komentarzami.

Zadanie 26.5. Z26_5.java

Mnożenie wielomianu przez liczbę różną od zera (liczby przez wielomian — działanie jest przemienne) polega na pomnożeniu wszystkich współczynników wielomianu przez tę liczbę, np. $2 \cdot (3x^2 - 2x + 4) = 6x^2 - 4x + 8$. Stopień wielomianu nie ulega przy tym zmianie. Jeśli liczba równa jest *0*, to w wyniku mnożenia otrzymamy wielomian zerowy.

```
/** Zadanie Z26.5 */
import java.util.Arrays;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Z26_5 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1};
        System.out.println("a = "+Arrays.toString(a));
        double b = -2;
        System.out.println("b = "+b);
        /* Iloczyn wielomianu przez liczbę (różną od zera) */
        double[] w = new double[a.length];
        for(int i = 0; i < a.length; ++i)
            w[i] = a[i]*b;
        System.out.println("Iloczyn wielomianu a przez liczbę b: ");
        System.out.println("a*b = "+Arrays.toString(w));
    }
}
```

Zadanie 26.6. Z26_6.java

Mnożąc dwa niezerowe wielomiany stopnia *m* i *n*, otrzymujemy niezerowy wielomian stopnia *m*+*n*. Mnożymy każdy wyraz jednego wielomianu przez każdy wyraz drugiego wielomianu i otrzymane jednomiany dodajemy (po uporządkowaniu jednomianów względem ich stopnia — postępujemy tak, wykonując obliczenia na kartce). W programie wygodnie będzie dodawać jednomiany na bieżąco, zaraz po ich obliczeniu (`w[i+j] += a[i]*b[j]`).

```
/** Zadanie Z26.6 */
import java.util.Arrays;
public class Z26_6 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1};
        System.out.println("a = "+Arrays.toString(a));
        double[] b = {3, -4, 2, 1, 2};
        System.out.println("b = "+Arrays.toString(b));
        /* Iloczyn wielomianów */
        double[] w = new double[a.length+b.length-1];
        for(int i = 0; i < a.length; ++i)
            for(int j = 0; j < b.length; ++j)
                w[i+j] += a[i]*b[j];
        System.out.println("Iloczyn wielomianów: ");
        System.out.println("a*b = "+Arrays.toString(w));
    }
}
```


Uwaga

Stopień wielomianu `a` jest równy `a.length-1`, stopień wielomianu `b` jest równy `b.length-1`, więc stopień iloczynu wielomianów wynosi `a.length+b.length-2`. Stąd wynika deklaracja `double[] w = new double[a.length+b.length-1]` — rozmiar tablicy jest o *1* większy od stopnia wielomianu.

Zadanie 26.7. Z26_7.java

Pochodna wielomianu jest wielomianem (stopnia o *1* niższego)

$$\left(a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0\right)' = n a_n x^{n-1} + (n-1)a_{n-1}x^n + \ldots + a_1 .$$

Ebookpoint.pl kopia dla: pawher@o2.pl

```
/** Zadanie Z26.7 */
import java.util.Arrays;
public class Z26_7 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1};
        System.out.println("a = "+Arrays.toString(a));
        /* Pochodna wielomianu */
        double[] w = new double[a.length-1];
        for(int i = 0; i < w.length; ++i)
            w[i] = a[i+1]*(i+1);
        System.out.println("Pochodna wielomianu: ");
        System.out.println("a\' = "+Arrays.toString(w));
    }
}
```

Zadanie 26.8. Z26_8.java

Całka nieoznaczona (funkcja pierwotna) wielomianu jest wielomianem (stopnia o *1* wyższego) $\int\left(a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0\right)dx = \frac{a_n}{n+1}x^{n+1} + \frac{a_{n-1}}{n}x^n + \ldots + a_0 + c$, gdzie *c* jest dowolną stałą. W programie przyjmiemy $c = 0$.

```
/** Zadanie Z26.8 */
import java.util.Arrays;
public class Z26_8 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1};
        System.out.println("a = "+Arrays.toString(a));
        /* Całka wielomianu (funkcja pierwotna) */
        double[] w = new double[a.length+1];
        for(int i = 0; i < a.length; ++i)
            w[i+1] = a[i]/(i+1);
        w[0] = 0; // może być dowolna stała c
        System.out.println("Całka (funkcja pierwotna) wielomianu: ");
        System.out.println("A = "+Arrays.toString(w));
    }
}
```

Zadanie 26.9. Polynomial.java, Z26_9.java

Na początek w klasie `Polynomial` umieścimy konstruktor z jednym parametrem, tablicą liczb typu `double`. Ponieważ użytkownik może wskazać jako parametr tablicę zawierającą wartość *0* będącą ostatnim elementem (lub nawet kilka końcowych elementów będzie miało wartość *0*), więc w konstruktorze wykorzystamy metodę `korekta()`, zdefiniowaną w rozwiązaniu zadania 26.4 (metodę tę włączamy do klasy).

W klasie `Polynomial` przesłaniamy dziedziczoną z klasy `java.lang.Object` metodę `toString()`; zwracany łańcuch znaków jest zgodny ze wskazówką z zadania 26.3.

```
import java.util.Arrays;
public class Polynomial {
    private double[] w;
    private static double[] korekta(double[] a) {
        int n = a.length;
        while (a[n-1] == 0 && n > 1) --n;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        return Arrays.copyOf(a, n);
    }
    public Polynomial(double[] a) {
        a = korekta(a);
        w = new double[a.length];
        for(int i = 0; i < a.length; ++i)
            this.w[i] = a[i];
    }
    @Override public String toString() {
        return Arrays.toString(this.w);
    }
}
```

Do klasy dołączymy kilka metod. Na podstawie rozwiązania zadania 26.1 budujemy publiczną metodę `horner()`, obliczającą wartość wielomianu reprezentowanego przez obiekt dla argumentu podanego jako parametr wywołania metody.

```
public double horner(double x) {
    double y = 0.0;
    for(int i = w.length-1; i >= 0; --i) {
        y *= x;
        y += this.w[i];
    }
    return y;
}
```

Metodę `add()`, która do wielomianu reprezentowanego przez wywołujący ją obiekt będzie dodawać wielomian reprezentowany przez parametr `a`, konstruujemy na podstawie rozwiązania zadania 26.3. Działanie wykonujemy na dwóch tablicach: `this.w` (wstawiamy w miejsce tablicy `a`) i `a.w` (w miejsce `b`).

```
public Polynomial add(Polynomial a) {
    double[] s = new double[Math.max(this.w.length, a.w.length)];
    int i = 0;
    for(i = 0; i < Math.min(this.w.length, a.w.length); ++i)
        s[i] = this.w[i]+a.w[i];
    if (this.w.length > a.w.length)
        while (i < this.w.length) {
            s[i] = this.w[i];
            ++i;
        }
    else if (this.w.length < a.w.length)
        while (i < a.w.length) {
            s[i] = a.w[i];
            ++i;
        }
    else if(s[s.length-1] == 0)
        korekta(s);
    return new Polynomial(s);
}
```

Podobnie skonstruujemy metodę `sub()`, która od wielomianu reprezentowanego przez wywołujący ją obiekt będzie odejmować wielomian reprezentowany przez parametr `a`. Pamiętajmy o uwadze (o zmianie znaków) z rozwiązania zadania 26.4.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public Polynomial sub(Polynomial a) {
    double[] s = new double[Math.max(this.w.length, a.w.length)];
    int i = 0;
    for(i = 0; i < Math.min(this.w.length, a.w.length); ++i)
        s[i] = this.w[i]-a.w[i];
    if (this.w.length > a.w.length)
        while (i < this.w.length) {
            s[i] = this.w[i];
            ++i;
        }
    else if (this.w.length < a.w.length)
        while (i < a.w.length) {
            s[i] = -a.w[i];
            ++i;
        }
    else if(s[s.length-1] == 0)
        korekta(s);
    return new Polynomial(s);
}
```

Na podstawie rozwiązania zadania 26.5 zbudujemy metodę `mult()`, która wielomian reprezentowany przez wywołujący ją obiekt będzie mnożyć przez liczbę typu `double` podaną jako parametr.

```
public Polynomial mult(double a) {
    double[] tmp = new double[w.length];
    for(int i = 0; i < w.length; ++i)
        tmp[i] = this.w[i]*a;
    return new Polynomial(tmp);
 }
```

Przeciążając metodę `mult()`, zrealizujemy mnożenie wielomianu przez wielomian (na podstawie rozwiązania zadania 26.6).

```
public Polynomial mult(Polynomial a) {
    double[] tmp = new double[this.w.length+a.w.length-1];
    for(int i = 0; i < this.w.length; ++i)
        for(int j = 0; j < a.w.length; ++j)
            tmp[i+j] += this.w[i]*a.w[j];
    return new Polynomial(tmp);
}
```

Na podstawie rozwiązania zadania 26.7 zbudujemy metodę `derivative()` (ang. *derivative* — pochodna), wyznaczającą pochodną wielomianu reprezentowanego przez obiekt wywołujący metodę.

```
public Polynomial derivative() {
    double[] tmp = new double[this.w.length-1];
    for(int i = 0; i < tmp.length; ++i)
        tmp[i] = this.w[i+1]*(i+1);;
    return new Polynomial(tmp);
}
```

Natomiast wzorując się na rozwiązaniu zadania 26.8, utworzymy metodę `integral()` (ang. *integral* — całka), wyznaczającą funkcję pierwotną wielomianu reprezentowanego przez obiekt.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public Polynomial integral() {
    double[] tmp = new double[this.w.length+1];
    for(int i = 0; i < this.w.length; ++i)
        tmp[i+1] = this.w[i]/(i+1);
    tmp[0] = 0;
    return new Polynomial(tmp);
}
```

Analizę wierszy programu demonstrującego możliwości klasy `Polynomial` pozostawiamy Czytelnikowi.

```
/** Zadanie Z26.9 */
import java.util.*;
public class Z26_9 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1, 0};
        Polynomial w1 = new Polynomial(a);
        System.out.println("w1 = "+w1);
        System.out.println(Arrays.toString(a));
        System.out.println("Wartości wielomianu w1(x) dla x z przedziału
             <0,5; 2,0> z krokiem 0,25.");
        double x = 0.5;
        while (x <= 2.0) {
            System.out.printf("w1(%4.2f) = %8.4f\n", x, w1.horner(x));
            x += 0.25;
        }
        double[] b = {2, -4, 3, -3, 5, 1};
        Polynomial w2 = new Polynomial(b);
        System.out.println("w1 = "+w1);
        System.out.println("w2 = "+w2);
        System.out.println("Suma w1+w2 = "+w1.add(w2));
        System.out.println("Suma w2+w1 = "+w2.add(w1));
        System.out.println("Różnica w1-w2 = "+w1.sub(w2));
        System.out.println("Różnica w2-w1 = "+w2.sub(w1));
        System.out.println("Iloczyn w1*2 = 2*w1 = "+w1.mult(2));
        System.out.println("Iloczyn w2*(-1) = (-1)*w2 = "+w2.mult(-1));
        System.out.println("Iloczyn w1*w2 = "+w1.mult(w2));
        System.out.println("Iloczyn w2*w1 = "+w2.mult(w1));
        System.out.println("Pochodna (w1)\' = "+w1.derivative());
        System.out.println("Pochodna (w2)\' = "+w2.derivative());
        System.out.println("Funkcja pierwotna dla w1: "+w1.integral());
        System.out.println("Funkcja pierwotna dla w2: "+w2.integral());
        System.out.println("Wielomian w1 = "+w1);
        System.out.println("Pochodna z funkcji pierwotnej (dla w1): "+
            w1.integral().derivative());
        System.out.println("Funkcja pierwotna z pochodnej (dla w1): "+
            w1.derivative().integral());
    }
}
```

Zadanie 26.10. Z26_10.java

Wielomian $w(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ możemy przedstawić w postaci

$$w(x) = (x-c)\left(b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + ... + b_0\right) + r =$$
$$= b_{n-1}x^n + \left(b_{n-2} - cb_{n-1}\right)x^{n-1} + \left(b_{n-3} - cb_{n-2}\right)x^{n-2} + ... + \left(b_0 - cb_1\right)x + r - cb_0 .$$

Ebookpoint.pl kopia dla: pawher@o2.pl

Porównując obie postacie wielomianu, uzyskamy:

$$b_{n-1} = a_n, b_{n-2} - cb_{n-1} = a_{n-1}, b_{n-3} - cb_{n-2} = a_{n-2}, ..., b_0 - cb_1 = a_1, r - cb_0 = a_0 .$$

Po uporządkowaniu wyrażeń otrzymamy ostateczną postać serii wzorów opisujących współczynniki wielomianu *b* i resztę z dzielenia *r*:

$$b_{n-1} = a_n, b_{n-2} = a_{n-1} + cb_{n-1}, b_{n-3} = a_{n-2} + cb_{n-2}, ..., b_0 = a_1 + cb_1, r = a_0 + cb_0 .$$

Podany algorytm dzielenia wielomianu przez dwumian *(x–c)* nazywamy *schematem Hornera* (dzielenia wielomianu przez dwumian).

```
/** Zadanie Z26.10 */
import java.util.Arrays;
public class Z26_10 {
    public static void main(String[] args) {
        double[] a = {-1, 4, 2, 1};
        System.out.println("Wielomian a = "+Arrays.toString(a));
        double c = -2;
        System.out.printf("Dwumian (x-c) = (x%+.2f)\n",(-c));
        /* Dzielenie wielomianu przez dwumian x-c */
        double[] w = new double[a.length-1];
        double r;
        int i = w.length-1;
        w[i] = a[i+1];
        while (i > 0) {
            w[i-1] = a[i]+w[i]*c;
            --i;
        }
        r = a[i]+w[i]*c;
        System.out.println("a:(x-c) = "+Arrays.toString(w)+", r = "+r);
    }
}
```

Zadanie 26.11. Z26_11.java

Na podstawie rozwiązania zadania 26.10 możemy utworzyć metodę `division()`, obliczającą iloraz wielomianu przez dwumian *(x–c)* i zwracającą wynik w postaci nowego obiektu `Polynomial`. Powstałą resztę z dzielenia w tym przypadku pomijamy.

```
public Polynomial division(int c) {
    double[] tmp = new double[this.w.length-1];
    int i = tmp.length-1;
    tmp[i] = this.w[i+1];
    while (i > 0) {
        tmp[i-1] = this.w[i]+tmp[i]*c;
        --i;
    }
    return new Polynomial(tmp);
}
```

Resztę z dzielenia wielomianu przez dwumian *(x–c)* zwróci nam metoda `remainder()`. Tym razem pomijamy uzyskany w trakcie obliczeń iloraz (wielomian `tmp`) i zwracamy jedynie resztę (liczbę typu `double`).

```
public double remainder(int c) {
    double[] tmp = new double[this.w.length-1];
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        int i = tmp.length-1;
        tmp[i] = this.w[i+1];
        while (i > 0) {
            tmp[i-1] = this.w[i]+tmp[i]*c;
            --i;
        }
        return this.w[i]+tmp[i]*c;
    }
```

Stosując obie zdefiniowane metody, możemy wykonywać *dzielenie z resztą* wielomianu przez dwumian o postaci *(x–c)*. Przedstawia to następujący przykład:

```
/** Zadanie Z26.11 */
import java.util.Arrays;
public class Z26_11 {
    public static void main(String[] args) {
        double[] a = {1, 0, 0, 1};
        Polynomial w = new Polynomial(a);
        System.out.println("Wielomian w = "+w);
        double c = -1;
        System.out.println("w:(x-c) = "+w.division(-1));
        System.out.println("r = "+w.remainder(-1));
    }
}
```

Zadanie 26.12. Z26_12.java

Zadanie zostało rozwiązane dla danych przykładowych. Kod potrzebny do wprowadzenia danych z klawiatury (stopnia wielomianu, tablicy współczynników `t` i granic całkowania `a` i `b`) pozostawiamy do napisania Czytelnikowi.

W rozwiązaniu zadania istotne są trzy kroki: utworzenie obiektu reprezentującego wielomian `Polynomial w = new Polynomial(t)`, wyznaczenie funkcji pierwotnej dla wielomianu `Polynomial f = w.integral()` i obliczenie wartości całki oznaczonej `double s = f.horner(b)-f.horner(a)`.

```
/** Zadanie Z26.12 */
public class Z26_12 {
    public static void main(String[] args) {
        /* Dane przykładowe */
        double[] t = {1, -3, 3, -2, 1};
        double a = -2;
        double b = 3;
        /* Tworzenie obiektów i obliczenia */
        Polynomial w = new Polynomial(t);
        System.out.println("Wielomian w = "+w);
        System.out.printf("Granice całkowania\na = %.2f\nb = %.2f\n",
            a, b);
        Polynomial f = w.integral();        // funkcja pierwotna
        double s = f.horner(b)-f.horner(a); // całka oznaczona
        System.out.printf("Całka oznaczona s = %.4f\n", s);
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 26.13. Z26_13.java

Jeśli liczby $x_1, x_2, ..., x_n$ są jedynymi pierwiastkami wielomianu, to wielomian możemy przedstawić w postaci iloczynu $w(x) = (x - x_1) \cdot (x - x_2) \cdot ... \cdot (x - x_n)$. Wykorzystamy ten wzór do wyznaczenia współczynników wielomianu w przypadku, gdy znane są jego pierwiastki.

Zaczniemy od zbudowania wielomianu $w(x) = 1$:

```
double[] tmp = {1};
Polynomial w = new Polynomial(tmp);
```

Następnie będziemy mnożyć ten wielomian przez dwumiany o postaci $(x - x_i)$ dla $i = 1, 2, \ldots, n$. Tworzymy tablicę dwuelementową `tmp`, która posłuży nam do budowania dwumianów o postaci *(x–c)*. Ustalamy współczynnik stojący przy *x* (`tmp[1] = 1`). Wprowadzamy z konsoli liczbę pierwiastków `n`. W pętli (`for(int i = 0; i < n; ++i)`) wczytujemy kolejne pierwiastki (`tmp[0] = -input.nextDouble()` — zwróćmy uwagę na zmianę znaku wczytanej wartości) i wykorzystujemy je do zbudowania obiektu `Polynomial` reprezentującego dwumian (`new Polynomial(tmp)`). Mnożymy wielomian `w` przez dwumian i podstawiamy wynik do zmiennej `w` (`w = w.mult(new Polynomial(tmp))`).

Po wyjściu z pętli wyświetlamy wynik.

```
/** Zadanie Z26.13 */
import java.util.Scanner;
public class Z26_13 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double[] tmp = {1};
        Polynomial w = new Polynomial(tmp);
        tmp = new double[2];
        tmp[1] = 1;
        System.out.print("Liczba pierwiastków, n = ");
        int n = input.nextInt();
        for(int i = 0; i < n; ++i) {
            System.out.print("Pierwiastek, x = ");
            tmp[0] = -input.nextDouble();
            w = w.mult(new Polynomial(tmp));
        }
        System.out.println("w = "+w);
    }
}
```

27. Obliczenia statystyczne

Zadanie 27.1. 27_1.java

Wyznaczamy element maksymalny i element minimalny, a następnie obliczamy różnicę między nimi.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
import java.util.Arrays;
public class Z27_1 {
    public static void main(String[] args) {
        System.out.println("Próbka:");
        double[] x = {1.35, 2.45, 2.05, 1.20, 2.15, 1.70, 1.45, 1.95,
                      2.00, 1.65, 1.65, 2.05, 1.75, 1.25, 2.25, 1.40};
        System.out.println(Arrays.toString(x));
        /* Element minimalny */
        double minX = x[0];
        for (double xi: x)
            if (xi < minX)
                minX = xi;
        System.out.println("Element minimalny: "+minX);
        /* Element maksymalny */
        double maxX = x[0];
        for (double xi: x)
            if (xi > maxX)
                maxX = xi;
        System.out.println("Element maksymalny: "+maxX);
        /* Rozstęp badanej cechy */
        double r = maxX-minX;
        System.out.println("Rozstęp badanej cechy: "+r);
    }
}
```

Element maksymalny i minimalny możemy wyznaczyć w jednej pętli. Zamiast instrukcji warunkowych i porównywania można zastosować metody klasy `Math`.

```
double minX = x[0];
double maxX = x[0];
for (double xi: x) {
    minX = Math.min(minX, xi);
    maxX = Math.max(maxX, xi);
}
```

Możemy również posortować tablicę w porządku niemalejącym i wtedy pierwszy element będzie elementem minimalnym, a ostatni maksymalnym.

```
Arrays.sort(x); // sortowanie tablicy
System.out.println("Element minimalny: "+x[0]);
System.out.println("Element maksymalny: "+x[x.length-1]);
System.out.println("Rozstęp badanej cechy: "+(x[x.length-1]-x[0]));
```

Zadanie 27.2. Z27_2.java

Ustalamy wartość początkową sumy elementów (`double sa = 0.0`) i w pętli dodajemy do sumy kolejne elementy próbki. Po zakończeniu sumowania dzielimy sumę przez liczbę elementów próbki i otrzymujemy wartość średniej arytmetycznej badanej próbki. Wynik zaokrąglamy do dwóch miejsc po przecinku.

```
double sa = 0.0;
for(double xi: x)
    sa += xi;
sa /= n;
System.out.println("Średnia arytmetyczna: "+zaokr(sa));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Rozwiązania zadań 27.2 – 27.19 będą zbudowane według schematu (XX — oznacza jedno- lub dwucyfrowy numer zadania):

```
import java.util.Arrays;
public class Z27_XX {
    public static double zaokr(double x) {
        double prec = 100.0;
        return (int)(x*prec+0.5)/prec;
    }

    public static void main(String[] args) {
        System.out.println("Próbka:");
        double[] x = {1.35, 2.45, 2.05, 1.20, 2.15, 1.70, 1.45, 1.95,
                      2.00, 1.65, 1.65, 2.05, 1.75, 1.25, 2.25, 1.40};
        System.out.println(Arrays.toString(x));
        int n = x.length; // liczba elementów

        /* Tu wstawimy rozwiązanie zadania. */

    }
}
```

W odpowiedziach do zadań podamy wyłącznie fragmenty kodu odpowiedzialne za rozwiązanie problemu zawartego w zadaniu.

Zaokrąglenie wyniku w tym i kolejnych zadaniach wykonujemy przy użyciu statycznej metody `zaokr()`:

```
public static double zaokr(double x) {
    double prec = 100.0;
    return (int)(x*prec+0.5)/prec;
}
```

Algorytm zaokrąglania liczb przedstawimy na dwóch przykładach:

x	x*100	x*100+0.5	(int)(x*100+0.5)	(int)(x*100+0.5)/100
2,7136	271,36	271,86	271	2,71
3,5481	354,81	355,31	355	3,55

Zadanie 27.3. 27_3.java

Najpierw obliczamy iloczyn wszystkich *n* elementów próbki — ustalamy początkową wartość iloczynu (`double sg = 1.0`) i następnie w pętli mnożymy tę wartość przez kolejne elementy (`xi`) próbki, przechowując w zmiennej `sg` kolejne obliczone iloczyny (`sg *= xi`). Wartości elementów próbki muszą być nieujemne.

Z iloczynu wszystkich elementów próbki obliczamy pierwiastek *n*-tego stopnia (`sg = Math.pow(sg, 1.0/n)`) i uzyskujemy średnią geometryczną próbki ($\overline{g} = \sqrt[n]{\prod_{i=1}^{n} x_i}$).

```
/* Średnia geometryczna - wzór nr 1 */
double sg = 1.0;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
for(double xi: x)
    sg *= xi;
sg = Math.pow(sg, 1.0/n);
System.out.println("Średnia geometryczna: "+zaokr(sg));
```

Jeśli wartości elementów próbki są dodatnie, to możemy zastosować własności logarytmów $\log \overline{g} = \log \sqrt[n]{\prod_{i=1}^{n} x_i} = \frac{1}{n} \log \prod_{i=1}^{n} x_i = \frac{1}{n} \sum_{i=1}^{n} \log x_i$, czyli średnia geometryczna $\overline{g} = e^{\frac{1}{n}\sum_{i=1}^{n} \log x_i}$.

```
/* Średnia geometryczna - wzór nr 2 */
double lnsg = 0;
for(double xi: x)
    lnsg += Math.log(xi);
lnsg /= n;
double sg = Math.exp(lnsg);
System.out.println("Średnia geometryczna: "+zaokr(sg));
```

Zadanie 27.4. 27_4.java

Średnia harmoniczna ciągu liczb jest odwrotnością średniej arytmetycznej odwrotności tych liczb. Wszystkie liczby muszą być różne od zera. Suma odwrotności również musi być różna od zera. Najpierw obliczamy sumę odwrotności tych liczb (`sh`), potem wyrażenie `sh/n` (średnią arytmetyczną odwrotności) i jego odwrotność `n/sh`, która jest ostatecznym wynikiem (`sh = n/sh`). Zwróćmy przy tym uwagę na wykorzystanie zmiennej `sh`, która ma w czasie obliczeń różne interpretacje.

```
double sh = 0.0;
for(double xi: x)
    sh += 1/xi;
sh = n/sh;
System.out.println("Średnia harmoniczna: "+zaokr(sh));
```

Zadanie 27.5. 27_5.java

```
/* Średnia potęgowa rzędu 2. */
double sp2 = 0.0;
for(double xi: x)
    sp2 += Math.pow(xi, 2);
sp2 = Math.pow(sp2/n, 1.0/2);
System.out.println("Średnia potęgowa rzędu 2.: "+zaokr(sp2));
```

Podstawienie `sp2 = Math.pow(sp2/n, 1.0/2);` można zastąpić równoważnym wzorem `sp2 = Math.sqrt(sp2/n);` (obliczanie pierwiastka kwadratowego).

```
/* Średnia potęgowa rzędu 3. */
double sp3 = 0.0;
for(double xi: x)
sp3 += Math.pow(xi, 3);
sp3 = Math.pow(sp3/n, 1.0/3);
System.out.println("Średnia potęgowa rzędu 3.: "+zaokr(sp3));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

W tym przykładzie również wyrażenie `Math.pow(sp3/n, 1.0/3)` można zastąpić wyrażeniem `Math.cbrt(sp3/n)` (obliczanie pierwiastka trzeciego stopnia). W ogólnym przypadku jest to wyrażenie o postaci `Math.pow(sp/n, 1.0/r)`, gdzie `r` oznacza rząd średniej potęgowej, a `sp` — jej wartość. Warto jednak (przy dużej ilości obliczeń) zastanowić się nad zastąpieniem metody obliczającej potęgi (`Math.pow()`) szybszym mnożeniem lub własną metodą — obliczającą potęgi o wykładniku całkowitym — o postaci:

```
static double power(double a, int n) {
    double p = 1.0;
    for(int i = 0; i< n; ++i)
        p *= a;
    return p;
}
```

Zadanie 27.6. 27_6.java

Medianę możemy wyznaczyć po posortowaniu próbki.

```
/* Sortowanie próbki */
Arrays.sort(x);
System.out.println("Posortowane:\n"+Arrays.toString(x));
```

Przy wyborze elementu środkowego należy pamiętać o rozbieżności pomiędzy indeksowaniem elementów próbki (od *1* do *n*) a indeksowaniem tablicy, rozpoczynającym się od *0*.

```
double me;
if (n%2 == 1)
    me = x[(n-1)/2]; // n jest nieparzyste, środek posortowanej próbki
else
    me = (x[n/2-1]+x[n/2])/2; // n parzyste, średnia wartości środkowych
System.out.println("Mediana: "+me);
```

Zadanie 27.7. 27_7.java

Zacznijmy od zdefiniowania struktury, która pozwoli nam na przechowywanie próbki w postaci *szeregu rozdzielczego punktowego*. W klasie `Dane` są dwa publiczne pola: `double x`, do przechowywania wartości elementu próbki, oraz `int n`, zawierające liczbę powtórzeń tej wartości.

```
class Dane {
    public
        double x;
        int n;
}
```

Próbkę wprowadzamy do tablicy:

```
double[] x = {1.35, 2.45, 2.05,1.35, 1.2, 1.35, 1.20, 2.15, 1.70, 1.45,
              1.95, 2.00, 1.65, 1.65, 2.05, 1.75, 1.35, 2.25, 1.65};
```

Tablicę sortujemy (należy importować klasę `java.util.Arrays`):

```
Arrays.sort(x);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Następnie musimy policzyć, ile różnych wartości zawiera próbka. Zewnętrzna pętla typu `while` pozwala na przejrzenie wszystkich wartości próbki (tablicy `x`), natomiast wewnętrzna pętla pomija kolejne, powtarzające się wartości (tablica `x` jest posortowana w kolejności niemalejącej). Licznik (zmienna `licznik`) jest inkrementowany tylko wtedy, gdy następny element jest różny od bieżącego.

```
int n = x.length;
int i = 0;
int licznik = 0;
while (i < n) {
    while (i < n-1 && x[i+1] == x[i])
        ++i;
        ++licznik;
    ++i;
}
```

Teraz zmienna `licznik` zawiera liczbę różnych wartości w badanej próbce. Możemy utworzyć tablicę o odpowiednim rozmiarze i zbudować szereg rozdzielczy punktowy.

```
Dane[] y = new Dane[licznik];
i = 0;
int j = 0;
while (i < n) {
    /* przechowanie kolejnej wartości próbki w szeregu rozdzielczym */
    y[j] = new Dane();
    y[j].x = x[i];
    /* liczenie powtórzeń tej wartości */
    y[j].n = 1;
    while (i < n-1 && x[i+1] == x[i]) {
        ++i;
        ++y[j].n;
    }
    /* następna wartość */
    ++j;
    ++i;
}
```

W tablicy `y` mamy uporządkowany niemalejąco, ze względu na wartość, szereg rozdzielczy. Pozostaje nam poszukiwanie dominanty. Zmienna `boolean jest` na koniec będzie zawierała informację, czy w szeregu jest dominanta (na początek zakładamy `jest = true`). Zmienna `dominanta` (typu `Dane`) będzie zawierała wartość dominanty (pole `x`), o ile ona istnieje, i liczbę jej wystąpień (pole `n`).

```
Dane dominanta = new Dane();
/* wartość początkowa, mniejsza od wszystkich wartości próbki */
dominanta.x = y[0].x - 1;
dominanta.n = 0;
boolean jest = true;
/* W pierwszym cyklu poniższej pętli, pierwszy element tablicy y
   zostanie uznany za dominantę. */
for(Dane w: y) {
    /* Jeśli liczba wystąpień kolejnej wartości jest równa liczbie
       wystąpień wartości uznawanej za dominantę, to dominanta nie
       istnieje. */
    if (w.n == dominanta.n)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            jest = false;
        /* Znaleziono kolejną wartość, która może być dominantą. */
        else if (w.n > dominanta.n) {
            dominanta.x = w.x;
            dominanta.n = w.n;
            jest = true;
        }
    }
```

Pozostaje sformułowanie ostatecznej odpowiedzi:

```
if (jest) {
    System.out.println("Dominanta D = "+dominanta.x);
    System.out.println("Liczba wystąpień dominanty: "+dominanta.n);
} else
    System.out.println("W próbce nie ma dominanty");
```

Zadanie 27.8. Z27_8.java

Średnią arytmetyczną `sa` badanej próbki obliczymy w sposób przedstawiony w zadaniu 27.2. Do obliczenia wariancji (dyspersji) zastosujemy kolejno podane wzory:

```
System.out.println("Obliczenie wariancji według wzoru:");
System.out.print("Wzór nr 1: ");
double s2 = 0.0;
for(double xi: x)
    s2 += Math.pow(xi-sa, 2);
s2 /= n;
System.out.println(s2);

System.out.print("Wzór nr 2: ");
s2 = 0.0;
for (double xi: x)
    s2 += Math.pow(xi, 2);
s2 /= n;
s2 -= sa*sa;
System.out.println(s2);

System.out.print("Wzór nr 3: ");
double a = 2.0; // dowolna liczba, np. 2.0
s2 = 0.0;
for(double xi: x)
    s2 += Math.pow(xi-a, 2);
s2 /= n;
s2 -= Math.pow(sa-a, 2);
System.out.println(s2);
```

Wykorzystaną do obliczania kwadratów liczb metodę `Math.pow()` warto zastąpić mnożeniem lub metodą `sqr()` zbudowaną samodzielnie:

```
static double sqr(double x) {
    return x*x;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 27.9. 27_9.java

Obliczymy średnią arytmetyczną próbki i wariancję s^2 (zmienna `s2`) na podstawie rozwiązania zadań 27.2 i 27.8 (jeden wybrany wzór) oraz odchylenie standardowe (pierwiastek kwadratowy z wariancji):

```
double s = Math.sqrt(s2);
System.out.println("Odchylenie standardowe: "+zaokr(s));
```

Zadanie 27.10. 27_10.java

Odchylenie przeciętne od stałej `a` obliczymy, sumując bezwzględne wartości odchyleń `xi` (elementów tablicy) od stałej wartości `a` i dzieląc otrzymaną sumę przez liczbę elementów próbki `x.length` (rozmiar tablicy).

```
double a = 2.0; // wartość stałej a
double d = 0.0;
for(double xi: x)
    d += Math.abs(xi-a);
d /= x.length;
System.out.print("Odchylenie przeciętne od stałej a = "+a);
System.out.println(" jest równe "+zaokr(d));
```

Ponieważ odchylenie przeciętne próbki (od różnych wartości) jest często obliczane, to zdefiniujemy metodę:

```
static double odchyleniePrzec(double[] x, double a) {
    double d = 0.0;
    for(double xi: x)
        d += Math.abs(xi-a);
    return d/x.length;
}
```

i zastosujemy ją do wykonania obliczeń.

```
System.out.print("Odchylenie przeciętne od stałej a = "+a);
System.out.println(" jest równe "+zaokr(odchyleniePrzec(x, a)));
```

Metodę tę możemy wykorzystać również w rozwiązywaniu kolejnych zadań — 27.11 i 27.12.

Zadanie 27.11. 27_11.java

Obliczymy średnią arytmetyczną `sa` próbki na podstawie rozwiązania zadania 27.2, a następnie odchylenie przeciętne `d1` od średniej arytmetycznej:

```
double d1 = 0.0;
for(double xi: x)
    d1 += Math.abs(xi-sa);
d1 /= n;
System.out.print("Odchylenie przeciętne od średniej arytmetycznej: ");
System.out.println(zaokr(d1));
```

Możemy też zastosować metodę `odchyleniePrzec()`, zbudowaną w rozwiązaniu zadania 27.10:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
System.out.print("Odchylenie przeciętne od średniej arytmetycznej: ");
System.out.println(zaokr(odchyleniePrzec(x, sa)));
```

Zadanie 27.12. 27_12.java

Obliczymy medianę `me` próbki na podstawie rozwiązania zadania 27.6, a następnie odchylenie przeciętne `d2` od mediany:

```
double d2 = 0.0;
for(double xi: x)
    d2 += Math.abs(xi-me);
d2 /= n;
System.out.print("Odchylenie przeciętne od mediany: ");
System.out.println(zaokr(d2));
```

Możemy też zastosować metodę `odchyleniePrzec()`, zbudowaną w rozwiązaniu zadania 27.10:

```
System.out.print("Odchylenie przeciętne od mediany: ");
System.out.println(zaokr(odchyleniePrzec(x, me)));
```

Zadanie 27.13. 27_13.java

Zaczniemy od posortowania próbki. Rozważając różne warianty długości próbki, wyznaczymy położenie mediany (drugiego kwartyla Q_2) lub indeksów elementów służących do jej wyznaczenia. Dzielimy próbkę na dwie części: elementy mniejsze od mediany i medianę oraz medianę i elementy większe od mediany. Dla pierwszej części wyznaczamy medianę — to będzie dolny kwartyl (inaczej: pierwszy kwartyl) Q_1. Górnym (trzecim) kwartylem Q_3 jest mediana drugiej części.

Związek pomiędzy kwartylami i wartościami próbki w posortowanej tablicy przedstawiono w tabeli.

Liczebność próbki	Q_1	Q_2	Q_3
$n = 4k$	$x\left[\frac{n}{4}\right]$	$\frac{x\left[\frac{n}{2}-1\right]+x\left[\frac{n}{2}\right]}{2}$	$x\left[\frac{3n}{4}-1\right]$
$n = 4k+1$	$x\left[\frac{n-1}{4}\right]$	$x\left[\frac{n-1}{2}\right]$	$x\left[\frac{3n-3}{4}\right]$
$n = 4k+2$	$\frac{x\left[\frac{n-2}{4}\right]+x\left[\frac{n+2}{4}\right]}{2}$	$\frac{x\left[\frac{n}{2}-1\right]+x\left[\frac{n}{2}\right]}{2}$	$\frac{x\left[\frac{3n-6}{4}\right]+x\left[\frac{3n-2}{4}\right]}{2}$
$n = 4k+3$	$\frac{x\left[\frac{n-3}{4}\right]+x\left[\frac{n+1}{4}\right]}{2}$	$x\left[\frac{n-1}{2}\right]$	$\frac{x\left[\frac{3n-5}{4}\right]+x\left[\frac{3n-1}{4}\right]}{2}$

```
import java.util.Arrays;
public class Z27_13 {
    static double zaokr(double x) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        double prec = 1000.0;
        return (int)(x*prec+0.5)/prec;
    }

    public static void main(String[] args) {
        double[] x = {1.35, 2.45, 2.35, 1.20, 2.15, 1.70, 1.45, 1.95,
                      2.00, 1.65, 1.85, 2.05, 1.75, 1.25, 2.25, 2.05};
        System.out.println("Próbka: \n"+Arrays.toString(x));
        /* Sortowanie */
        Arrays.sort(x);
        System.out.println("Próbka posortowana: \n"+Arrays.toString(x));
        int n = x.length;
        double Q1 = 0.0, Q3 = 0.0;
        switch (n%4) {
            case 0:
                Q1 = x[n/4];
                Q3 = x[3*n/4-1];
            break;
            case 1:
                Q1 = x[(n-1)/4];
                Q3 = x[(3*n-3)/4];
            break;
            case 2:
                Q1 = (x[(n-2)/4]+x[(n+2)/4])/2;
                Q3 = (x[(3*n-6)/4]+x[(3*n-2)/4])/2;
            break;
            case 3:
                Q1 = (x[(n-3)/4]+x[(n+1)/4])/2;
                Q3 = (x[(3*n-5)/4]+x[(3*n-1)/4])/2;
            break;
        }
        double Q = (Q3-Q1)/2;
        System.out.println("Kwartyl dolny Q1: "+Q1);
        System.out.println("Kwartyl górny Q3: "+Q3);
        System.out.println("Odchylenie ćwiartkowe: "+Q);

    }
}
```

Zadanie 27.14. 27_14.java

Ze względu na dużą liczbę obliczanych potęg o wykładniku naturalnym wykorzystamy metodę `power()` z rozwiązania zadania 27.6.

```
/* Moment zwykły m2 rzędu 2. */
double m2 = 0.0;
for(double xi: x)
    m2 += power(xi, 2);
m2 /= n;
System.out.print("Moment zwykły m2 rzędu 2.");
System.out.println(" jest równy: "+zaokr(m2));
```

Powyższy fragment kodu możemy skopiować i zmodyfikować w celu obliczenia *momentu zwykłego* `m3` (*trzeciego rzędu*) i `m4` (*czwartego rzędu*). Wygodniej jednak będzie na podstawie tego kodu zbudować metodę obliczającą moment zwykły rzędu podanego jako drugi parametr:

Ebookpoint.pl kopia dla: pawher@o2.pl

```
static double momentZw(double[] x, int rz) {
    double m = 0.0;
    for(double xi: x)
        m += power(xi, rz);
    return m/x.length;
}
```

i w pętli obliczyć żądane momenty zwykłe:

```
/* Momenty zwykłe 2., 3. i 4. rzędu */
for(int r=2; r < 5; ++r) {
    System.out.printf("Moment zwykły rzędu "+r);
    System.out.println(" jest równy "+zaokr(momentZw(x, r)));
}
```

Zadanie 27.15. 27_15.java

W sposób znany z poprzednich zadań najpierw obliczamy średnią arytmetyczną `sa` badanej próbki, a następnie *moment centralny zwykły*. W obliczeniach wykorzystamy metodę `power()` z zadania 27.6.

```
/* Moment centralny M2 rzędu 2. */
double M2 = 0.0;
for(double xi: x)
    M2 += power(xi-sa, 2);
M2 /= n;
System.out.print("Moment centralny M2 rzędu 2.");
System.out.println(" jest równy: "+zaokr(M2));
```

W podobny sposób obliczymy momenty centralne `M3` (trzeciego rzędu) i `M4` (czwartego rzędu).

Zamiast takiego rozwiązania proponujemy metodę:

```
static double momentCent(double[] x, int rz, double sa) {
    double m = 0.0;
    for(double xi: x)
        m += power(xi-sa, rz);
    return m/x.length;
}
```

i obliczenia w pętli:

```
/* Momenty centralne 2., 3. i 4. rzędu */
for(int r=2; r < 5; ++r) {
    System.out.printf("Moment centralny rzędu "+r);
    System.out.println(" jest równy "+zaokr(momentCent(x, r, sa)));
}
```

Ostatni (trzeci) parametr metody `momentCent()` jest średnią arytmetyczną próbki. Możemy zbudować metodę bez tego parametru, która obliczy najpierw średnią arytmetyczną, a potem moment centralny zwykły.

```
static double momentCent(double[] x, int rz) {
    double sa = 0.0; // średnia arytmetyczna
    for(double xi: x)
        sa += xi;
    sa /= x.length;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    double m = 0.0;  // moment centralny zwykły
    for(double xi: x)
        m += power(xi-sa, rz);
    return m/x.length;
}
```

W tym przypadku nie będzie potrzebne wcześniejsze liczenie średniej arytmetycznej w programie.

```
for(int r=2; r < 5; ++r) {
    System.out.printf("Moment centralny rzędu "+r);
    System.out.println(" jest równy "+zaokr(momentCent(x, r)));
}
```

Obie metody (z dwoma i trzema parametrami) mogą istnieć w tej samej klasie, pod jedną (przeciążoną) nazwą `momentCent`.

Zadanie 27.16. 27_16.java

Zadanie rozwiązujemy w sposób analogiczny do zadania 27.14. Porównując wzór na *moment zwykły rzędu l* $m_l = \frac{1}{n}\sum_{i=1}^{n} x_i^l, \quad l \in N$ ze wzorem na *moment absolutny zwykły rzędu l* $a_l = \frac{1}{n}\sum_{i=1}^{n} |x_i|^l, \quad l \in N$, dostrzeżemy różnicę, którą należy uwzględnić w kodzie:

```
for(double xi: x)
    a += power(Math.abs(xi), rz);
```

Zadanie 27.17. 27_17.java

Zadanie rozwiązujemy w sposób analogiczny do zadania 27.15. Porównując wzór na *moment centralny rzędu l* $M_l = \frac{1}{n}\sum_{i=1}^{n} (x_i - \bar{x})^l, \quad l \in N$ ze wzorem na *moment absolutny centralny rzędu l* $b_l = \frac{1}{n}\sum_{i=1}^{n} |x_i - \bar{x}|^l, \quad l \in N$, dostrzeżemy różnicę, którą należy uwzględnić w kodzie. Zamiast:

```
for(double xi: x)
    m += power(xi-sa, rz);
```

napiszemy:

```
for(double xi: x)
    a += power(Math.abs(xi-sa), rz);
```

Zadanie 27.18. 27_18.java

Obliczamy kolejno średnią arytmetyczną `sa` (27.2), wariancję `s2` (27.8) i odchylenie standardowe `s` (27.9). Na tej podstawie wyznaczymy:

```
/* Współczynnik zmienności v */
double v = s/sa*100;
System.out.println("Współczynnik zmienności: "+zaokr(v)+"%");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 27.19. 27_19.java

Obliczamy średnią arytmetyczną `sa` i odchylenie przeciętne od średniej arytmetycznej `d1`.

```
/* Współczynnik nierównomierności H */
double h = d1/sa*100;
System.out.println("Współczynnik nierównomierności: "+zaokr(h)+"%");
```

Zadanie 27.20. Stat.java, Z27_20.java

W klasie `Stat` (`public class Stat {...}`) umieścimy zestaw metod statycznych wykonujących obliczenia na elementach tablicy jednowymiarowej `double[] x` (próbki) przekazanej jako parametr.

Zaczniemy od metody `zaokr()` z jednym parametrem `x`, zaokrąglającej przekazany argument do dwóch miejsc po przecinku i zwracającej zaokrąglony wynik.

```
public static double zaokr(double x) {
    double prec = 100.0;
    return (int)(x*prec+0.5)/prec;
}
```

Przeciążając metodę `zaokr()`, dodamy drugi parametr `n` — liczbę całkowitą wyrażającą liczbę miejsc po przecinku w wyniku.

```
public static double zaokr(double x, int n) {
    double prec = Math.pow(10, n);
    return (int)(x*prec+0.5)/prec;
}
```

Na podstawie rozwiązania zadania 27.1 zbudujemy metody `minX()` i `maxX()` wyznaczające wartość *minimalną* i *maksymalną* z badanej próbki (tablicy).

```
public static double minX(double[] x) {
    double min = x[0];
    for(double xi: x)
        if (xi < min) min = xi;
    return min;
}
public static double maxX(double[] x) {
    double max = x[0];
    for(double xi: x)
        if (xi > max) max = xi;
    return max;
}
```

Mając wartość minimalną i maksymalną, możemy wyznaczyć rozstęp.

```
public static double rozstX(double[] x) {
    return maxX(x)-minX(x);
}
```

Kolejna metoda oblicza średnią arytmetyczną (zob. zadanie 27.2).

```
public static double srednArytm(double[] x) {
    double sa = 0.0;
    for(double xi: x)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            sa += xi;
        return sa/x.length;
    }
```

W ten sposób na podstawie rozwiązanych zadań od 27.3 do 27.19 zbudujemy kolejne metody statyczne. Zredagowanie komentarzy dokumentacyjnych i utworzenie dokumentacji klasy pozostawiamy Czytelnikowi (zob. rozdział 19., „Dokumentacja klasy").

W przykładzie pokazano rozwiązania zadań 27.1 i 27.2 wykorzystujące statyczne metody z klasy `Stat`.

```
import java.util.Arrays;
public class Z27_20 {
    public static void main(String[] args) {
        System.out.println("Próbka:");
        double[] x = {1.35, 2.45, 2.05, 1.20, 2.15, 1.70, 1.45, 1.95,
                      2.00, 2.10, 1.50, 2.05};
        System.out.println(Arrays.toString(x));
        System.out.println("Wartość minimalna: "+Stat.minX(x));
        System.out.println("Wartość maksymalna: "+Stat.maxX(x));
        System.out.print("Rozstęp badanej cechy w próbce: ");
        System.out.println(Stat.zaokr(Stat.rozstX(x)));
        System.out.print("Średnia arytmetyczna: ");
        System.out.println(Stat.zaokr(Stat.srednArytm(x)));
    }
}
```

Zadanie 27.21. Statpr.java, Z27_21.java

Klasa `Statpr` zawiera jedno prywatne pole `pr` będące referencją do tablicy `double[]` zawierającej badaną próbkę i konstruktor.

```
import java.util.Arrays;
public class Statpr {
    private
       double[] pr;
    public Statpr(double[] x) {
        pr = new double[x.length];
        pr = Arrays.copyOf(x, x.length);
    }
}
```

Konstruktor tworzy obiekt zawierający tablicę o rozmiarze identycznym z rozmiarem tablicy przekazanej jako parametr (badana próbka) i kopiuje zawartość parametru do obiektu.

Do klasy kopiujemy kody metod `zaokr()` (zob. rozwiązanie zadania 27.20).

```
public static double zaokr(double x) {
    double prec = 100.0;
    return (int)(x*prec+0.5)/prec;
}
public static double zaokr(double x, int n) {
    double prec = Math.pow(10, n);
    return (int)(x*prec+0.5)/prec;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Metody statyczne z klasy `Stat` (zob. rozwiązanie zadania 27.20 lub rozwiązania zadań od 27.1 do 27.19) przekształcamy na metody działające na polu `pr` obiektu klasy `Statpr`.

```
public double minX() {
    double min = pr[0];
    for(double xi: pr)
        if (xi < min) min = xi;
    return min;
}

public double maxX() {
    double max = pr[0];
    for(double xi: pr)
        if (xi > max) max = xi;
        return max;
    }

public double rozstX() {
    return this.maxX()-this.minX();
}

public double srednArytm() {
    double sa = 0.0;
    for(double xi: pr)
        sa += xi;
    return sa/pr.length;
}
```

W ten sam sposób budujemy dalsze metody. Nie należy zapomnieć o przesłonięciu metody `toString()` z klasy `Object` (metoda ta ułatwi wyświetlanie danych w konsoli):

```
@Override public String toString() {
    return Arrays.toString(pr);
}
```

Komentarze dokumentacyjne i opracowanie dokumentacji pozostawiamy Czytelnikowi (zob. rozdział 19.).

W przykładzie pokazano rozwiązania zadań 27.1 i 27.2 wykorzystujące obiekt `a` klasy `Stat`.

```
public class Z27_21 {
    static double[] x = {1.35, 2.45, 2.05, 1.20, 2.15, 1.70, 1.45, 1.95,
                         2.00, 2.10, 1.50, 2.05};
    public static void main(String[] args) {
        Statpr a = new Statpr(x);
        System.out.println("Próbka:\n"+a);
        System.out.println("Wartość minimalna: "+a.minX());
        System.out.println("Wartość maksymalna: "+a.maxX());
        System.out.print("Rozstęp badanej cechy w próbce: ");
        System.out.println(Statpr.zaokr(a.rozstX()));
        System.out.print("Średnia arytmetyczna: ");
        System.out.println(Statpr.zaokr(a.srednArytm()));
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

28. Tablice wielowymiarowe i macierze

Zadanie 28.1. Z28_1.java

Deklarujemy tablicę liczb całkowitych o trzech wierszach i dziesięciu kolumnach (`int[][] a = new int[3][10]`). Wiersze mają indeksy *0*, *1* i *2*, a kolumny *0*, *1*, *2*, …, *9*. W pętli `for` indeks `i` zmienia się od *0* do *9*, co daje nam kolejno dostęp do kolumn tablicy. W pierwszym wierszu podstawiamy `a[0][i] = i+1` (liczby o *1* większe od indeksu, czyli *1*, *2*, …, *10*), w drugim `a[1][i] = a[0][i]*a[0][i]` (kwadraty liczb z pierwszego wiersza) i w trzecim `a[2][i] = a[1][i]*a[0][i]` (iloczyn pierwszego i drugiego wiersza, czyli sześcian pierwszego wiersza).

Tablica dwuwymiarowa jest po prostu tablicą tablic (tablicą trzech wierszy, a każdy wiersz jest tablicą dziesięciu liczb całkowitych). Przekonamy się o tym podczas wyświetlania zawartości tablicy dwuwymiarowej na ekranie:

```
for(int[] x: a)
    System.out.println(Arrays.toString(x));
```

W pętli `for` pobieramy z tablicy `a` jednowymiarowe tablice `x` (typu `int[]`), które wyświetlamy w konsoli po ich zamianie na łańcuch znaków (`Arrays.toString(x)`).

```
import java.util.Arrays;
public class Z28_1 {
    public static void main(String[] args) {
        int[][] a = new int[3][10];
        for(int i = 0; i < 10; ++i) {
            a[0][i] = i+1;
            a[1][i] = a[0][i]*a[0][i];
            a[2][i] = a[1][i]*a[0][i];
        }
        for(int[] x: a)
            System.out.println(Arrays.toString(x));
    }
}
```

Wynik wyświetlany jest w postaci:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Zamiast zamieniać tablicę `x` na łańcuch znaków, możemy ponownie zastosować pętlę `for` i uzyskać dostęp do jej elementów. Zagnieżdżone pętle `for` i metoda `printf()` pozwolą na przejrzyste wyświetlenie tablicy dwuwymiarowej.

```
for(int[] x: a) {
    for(int n: x)
        System.out.printf("%5d", n);
    System.out.println();
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Inną możliwością, chyba najbardziej rozpowszechnioną, jest wykorzystanie dostępu do elementów tablicy przy użyciu indeksów:

```
for(int i = 0; i < a.length; ++i) {
    for(int j = 0; j < a[i].length; ++j)
        System.out.printf("%5d", a[i][j]);
    System.out.println();
}
```


Uwaga

Tablica jest obiektem i pole `length` zawiera rozmiar tablicy:

- `a.length` — liczba wierszy tablicy dwuwymiarowej `a`,
- `a[i].length` — liczba elementów *i*-tego wiersza tablicy `a` (liczba kolumn).

W tych dwóch przypadkach otrzymany rezultat przedstawia się następująco:

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

Zadanie 28.2. Z28_2.java

Deklarujemy tablicę dwuwymiarową o dziesięciu wierszach, nie podając liczby kolumn (`int[][] a = new int[10][]`). W pętli deklarujemy każdy wiersz oddzielnie, ustalając rozmiar (liczbę kolumn) o *1* większy od indeksu wiersza (`a[i] = new int[i+1]`). Po zadeklarowaniu wiersza wypełniamy go kolejnymi liczbami naturalnymi (`for(int j = 0; j <= i; ++j) a[i][j] = n++;`).

Wyświetlanie tablicy zrealizujemy przy wykorzystaniu dwóch zagnieżdżonych pętli `for` i metody `printf()`. Do obliczenia sum użyjemy również dwóch pętli; w wewnętrznej pętli będziemy sumować liczby w wierszu, a w zewnętrznej obliczymy sumę wierszy, czyli sumę wszystkich liczb z tablicy.

```
public class Z28_2 {
    public static void main(String[] args) {
        /* Deklaracja i wypełnienie tablicy */
        int[][] a = new int[10][];
        int n = 1;
        for(int i = 0; i < 10; ++i) {
            a[i] = new int[i+1];
            for(int j = 0; j <= i; ++j)
                a[i][j] = n++;
        }
        /* Wyświetlanie tablicy */
        for(int i = 0; i < a.length; ++i) {
            for(int j = 0; j < a[i].length; ++j)
                System.out.printf("%5d", a[i][j]);
            System.out.println();
        }
        /* Sumowanie liczb */
        int suma = 0;
        for(int i = 0; i < a.length; ++i) {
            int wiersz = 0;
            for(int j = 0; j < a[i].length; ++j)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                wiersz += a[i][j];
            System.out.printf("Suma liczb w %d. wierszu : %d\n",
                i, wiersz);
            suma += wiersz;
        }
        System.out.printf("Suma liczb w tablicy: %d\n", suma);
    }
}
```

Zadanie 28.3. Z28_3.java

W klasie `TInt` do wprowadzania danych do tablicy wykorzystamy obiekt `cin` (ang. *console input*) klasy `Scanner`. Obiekt `cin` zdefiniowano w klasie jako prywatne i statyczne pole; jest on dostępny dla wszystkich metod tej klasy.

W wersji podstawowej metoda `input()` posiada jeden parametr — dwuwymiarową tablicę liczb całkowitych (`int[][] x`). Zewnętrzna pętla `for` wskazuje kolejne wiersze tablicy, pętla wewnętrzna wskazuje elementy w wierszu. W konsoli jest wyświetlany znak zachęty w postaci łańcucha `?[i][j] =` (gdzie `i` — numer wiersza, `j` — numer kolumny wprowadzanego elementu tablicy) i użytkownik może wprowadzić wartość elementu tablicy (`x[i][j] = cin.nextInt()`).

Metoda `print()` również ma jeden parametr — dwuwymiarową tablicę liczb całkowitych. W zagnieżdżonych pętlach `for` pobieramy i wyświetlamy elementy tablicy oddzielone odstępami. Po wyświetleniu ostatniego elementu w wierszu przechodzimy na początek nowej linii.

```
import java.util.Scanner;
public class TInt {
    private static Scanner cin = new Scanner(System.in);

    public static void input(int[][] x) {
        for(int i = 0; i < x.length; ++i)
            for(int j = 0; j < x[i].length; ++j) {
                System.out.printf("?[%d][%d] = ", i, j);
                x[i][j] = cin.nextInt();
            }
    }

    public static void print(int[][] x) {
        for(int i = 0; i < x.length; ++i) {
            for(int j = 0; j < x[i].length; ++j)
                System.out.printf("%d ", x[i][j]);
            System.out.println();
        }
    }
}
```

Na podstawie metody `input()` z jednym parametrem warto zbudować metody z dwoma parametrami — pierwszy parametr jest znakiem (`char`) lub łańcuchem znaków (`String`), drugi parametr jest wczytywaną tablicą. W pierwszym parametrze przekazujemy nazwę tablicy (znak lub łańcuch znaków). Ta informacja będzie wyświetlana podczas wprowadzania danych, np. `a[0][1] =` lub `tablica[2][3] =`. Poprawi to czytelność programu.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public static void input(char name, int[][] x) {
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[i].length; ++j) {
            System.out.printf("%c[%d][%d] = ", name, i, j);
            x[i][j] = cin.nextInt();
        }
}

public static void input(String name, int[][] x) {
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[i].length; ++j) {
            System.out.printf("%s[%d][%d] = ", name, i, j);
            x[i][j] = cin.nextInt();
        }
}
```

Modyfikacja kodu metod `input()` pozwoli na zastosowanie ich do wczytywania tablic jednowymiarowych.

```
public static void input(int[] x) {
    for(int i = 0; i < x.length; ++i) {
        System.out.printf("?[%d] = ", i);
        x[i] = cin.nextInt();
    }
}

public static void input(char name, int[] x) {
        for(int i = 0; i < x.length; ++i) {
            System.out.printf("%c[%d] = ", name, i);
            x[i] = cin.nextInt();
        }
    }

public static void input(String name, int[] x) {
    for(int i = 0; i < x.length; ++i) {
        System.out.printf("%s[%d] = ", name, i);
        x[i] = cin.nextInt();
    }
}
```

Również metodę `print()` dostosujemy do wyświetlania tablic jednowymiarowych.

```
public static void print(int[] x) {
    for(int i = 0; i < x.length; ++i)
        System.out.printf("%d ", x[i]);
    System.out.println();
}
```

W przykładowej aplikacji wczytujemy i wyświetlamy zawartość dwuwymiarowych tablic `a` i `b` oraz tablicy jednowymiarowej, będącej częścią (wierszem) tablicy dwuwymiarowej.

```
public class Z28_3 {
    public static void main(String[] args) {
        int[][] a = new int[2][3];
        TInt.input(a);
        TInt.print(a);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        int[][] b = new int[2][2];
        TInt.input('b', b);
        TInt.print(b);
        TInt.input("a[1]", a[1]);
        TInt.print(a[1]);
    }
}
```

Zadanie 28.4. Z28_4.java

W klasie `Tint` importujemy dodatkowo klasę `Random` (`import java.util.Random;`) oraz tworzymy prywatny statyczny obiekt `rnd` (`private static Random rnd = new Random();`). Obiekt `rnd` i metodę `nextInt()` z klasy `Random` wykorzystamy do losowania wartości elementów tablicy.

```
public static void setRandom(int[][] x, int n) {
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[i].length; ++j)
            x[i][j] = rnd.nextInt(n+1);
}
```

Wylosowane liczby całkowite są nieujemne i mniejsze od *n+1*, czyli należą do przedziału $\langle 0, n \rangle$. Podobnie budujemy metodę wypełniającą wylosowanymi liczbami tablicę jednowymiarową.

```
public static void setRandom(int[] x, int n) {
    for(int i = 0; i < x.length; ++i)
        x[i] = rnd.nextInt(n+1);
}
```

W przykładowym programie wypełniamy liczbami całkowitymi pseudolosowymi z zakresu od *0* do *100* tablicę dwuwymiarową `a`. Tablicę jednowymiarową `b` wypełniamy liczbami z zakresu od *0* do *5*.

```
public class Z28_4 {
    public static void main(String[] args) {
        int[][] a = new int[5][4];
        TInt.setRandom(a, 100);
        TInt.print(a);
        int[] b = new int[10];
        TInt.setRandom(b, 5);
        TInt.print(b);
    }
}
```

Zadanie 28.5. Z28_5.java

Podczas wyświetlania tablic wielowymiarowych korzystne będzie wypisywanie liczb w kolumnach o ustalonej szerokości. Format wyświetlanych liczb przekazujemy jako pierwszy parametr, np. `"%d "` (liczba znaków zależy od wartości wyświetlanej liczby; po liczbie dodajemy jeden odstęp), `"%10d"` (stała szerokość pola, 10 znaków).

```
public static void printf(String format, int[][] x) {
    for(int i = 0; i < x.length; ++i) {
        for(int j = 0; j < x[i].length; ++j)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            System.out.printf(format, x[i][j]);
        System.out.println();
    }
}

public static void printf(String format, int[] x) {
    for(int i = 0; i < x.length; ++i)
        System.out.printf(format, x[i]);
    System.out.println();
}
```

Demonstrując działanie metody `printf()`, wyświetlamy w kolumnach o szerokości pięciu znaków tablicę dwuwymiarową o pięciu wierszach i czterech kolumnach, wypełnioną pseudolosowymi wartościami z zakresu od *0* do *100*.

```
public class Z28_5 {
    public static void main(String[] args) {
        int[][] a = new int[5][4];
        TInt.setRandom(a, 100);
        TInt.printf("%5d", a);
    }
}
```

Zadanie 28.6. Z28_6.java

Skopiujemy zawartość pliku *TInt.java* do pliku *TDouble.java*. Poprawimy nazwę klasy `TInt` na `TDouble`. W nagłówkach metod zamienimy typ parametru `int[][]` na `double[][]` (oraz `int[]` na `double[]`). W kodzie metod `input` zmienimy `cin.nextInt()` na `cin.nextDouble()`. W metodach `setRandom()` zmienimy drugi parametr `int n` na `double y` oraz wyrażenie losujące `rnd.nextInt(n+1)` na `y*rnd.nextDouble()`. W ten sposób przekształciliśmy klasę `TInt` na klasę `TDouble`, ułatwiającą pobieranie i wypisywanie danych oraz losowe ustawianie wartości w jedno- i dwuwymiarowych tablicach liczb typu `double`.

Demonstrując możliwości klasy, wypełnimy liczbami z przedziału $\langle 0;10{,}0\rangle$ tablicę dwuwymiarową i wyświetlimy jej zawartość w kolumnach, które będą miały szerokość ośmiu znaków i będą zawierać liczby z trzema miejscami po przecinku.

```
public class Z28_6 {
    public static void main(String[] args) {
        double[][] a = new double[5][4];
        TDouble.setRandom(a, 10.0);
        TDouble.printf("%8.3f", a);
    }
}
```

Zadanie 28.7. Z28_7.java

Zdefiniujemy metodę `sum()` dodającą dwie macierze (tablice dwuwymiarowe) i zwracającą wynik w takiej samej postaci. Sprawdzamy zgodność wymiarów macierzy. Jeśli wymiary nie są identyczne, to metoda rzuca wyjątek (`"Niezgodne wymiary macierzy"`). Tworzymy tablicę `tmp` o wymiarach takich samych jak parametry `x` i `y`. W zagnieżdżonych pętlach `for` dodajemy elementy tablic (`tmp[i][j] = x[i][j]+y[i][j]`). Jako wynik zwracamy tablicę `tmp`.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public static double[][] sum (double[][] x, double[][] y) {
    if (x.length != y.length || x[0].length != y[0].length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    double[][] tmp = new double[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[i].length; ++j)
            tmp[i][j] = x[i][j]+y[i][j];
    return tmp;
}
```

Demonstrując działanie metody `sum()`, dodamy dwie macierze wypełnione wylosowanymi wartościami typu `double`.

```
public class Z28_7 {
    /* Tu wstaw kod metody sum(). */
    public static void main(String[] args) {
        double[][] a = new double[5][4];
        TDouble.setRandom(a, 5.7);
        double[][] b = new double[5][4];
        TDouble.setRandom(b, 10.0);
        double[][] c = sum(a, b);
        System.out.println("Macierz A");
        TDouble.printf("%8.3f", a);
        System.out.println("Macierz B");
        TDouble.printf("%8.3f", b);
        System.out.println("Suma macierzy C = A+B");
        TDouble.printf("%8.3f", c);
    }
}
```


Uwaga

Na podstawie zdefiniowanej metody `sum()` możemy zbudować podobną metodę dla tablic dwuwymiarowych z elementami całkowitymi lub dla tablic jednowymiarowych.

Zadanie 28.8. Z28_8.java

Odejmowanie macierzy przedstawimy na przykładzie macierzy o elementach całkowitych. Wzorując się na rozwiązaniu zadania 28.7, utworzymy metodę `difference()` obliczającą różnicę macierzy.

```
public static int[][] difference (int[][] x, int[][] y) {
    if (x.length != y.length || x[0].length != y[0].length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    int[][] tmp = new int[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[i].length; ++j)
            tmp[i][j] = x[i][j]-y[i][j];
    return tmp;
}
```

Testując działanie metody `difference()`, obliczymy różnicę dwóch macierzy o elementach całkowitych, ustalonych w sposób losowy. Metodę `setRandom()` z klasy `TInt` możemy zastąpić metodą `input()` i wprowadzić wyrazy macierzy z konsoli.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
public class Z28_8 {
    /* Tu wstaw kod metody difference(). */
    public static void main(String[] args) {
        int[][] a = new int[3][6];
        TInt.setRandom(a, 150);
        int[][] b = new int[3][6];
        TInt.setRandom(b, 100);
        int[][] c = difference(a, b);
        System.out.println("Macierz A");
        TInt.printf("%5d", a);
        System.out.println("Macierz B");
        TInt.printf("%5d", b);
        System.out.println("Różnica macierzy C = A-B");
        TInt.printf("%5d", c);
    }
}
```

Zadanie 28.9. Z28_9.java

Iloczyn dwóch macierzy możemy policzyć tylko wtedy, gdy liczba kolumn pierwszej macierzy (`x[0].length`) jest równa liczbie wierszy drugiej macierzy (`y.length`), w przeciwnym wypadku metoda powinna zgłosić wyjątek. Wynik jest macierzą o liczbie wierszy równej liczbie wierszy pierwszej macierzy (`x.length`) i liczbie kolumn równej liczbie kolumn drugiej macierzy (`y[0].length`):

```
int[][] tmp = new int[x.length][y[0].length];
```

W dwóch zagnieżdżonych pętlach obliczymy wszystkie elementy macierzy (tablicy) będącej iloczynem. Zgodnie ze wzorem $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$ obliczamy w kolejnej pętli sumę `m` (`y.length`) iloczynów elementów z wiersza `i` pierwszej macierzy przez elementy kolumny `j` drugiej macierzy.

```
for(int k = 0; k < y.length; ++k)
    tmp[i][j] += x[i][k]*y[k][j];
```

Oto kompletny kod metody mnożącej dwie macierze:

```
public static int[][] product (int[][] x, int[][] y) {
    if (x[0].length != y.length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    int[][] tmp = new int[x.length][y[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < y[0].length; ++j)
            for(int k = 0; k < y.length; ++k)
                tmp[i][j] += x[i][k]*y[k][j];
    return tmp;
}
```

Dla macierzy (tablic) typu `double` wystarczy w kodzie zamienić typ tablic z `int` na `double`. Demonstrując działanie metody, pomnożymy dwie przykładowe macierze (wypełnione losowymi wartościami).

```
public class Z28_9 {
    /* Tu wstaw kod metody product(). */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        int[][] a = new int[2][3];
        TInt.setRandom(a, 15);
        int[][] b = new int[3][6];
        TInt.setRandom(b, 10);
        int[][] c = product(a, b);
        System.out.println("Macierz A");
        TInt.printf("%3d", a);
        System.out.println("Macierz B");
        TInt.printf("%3d", b);
        System.out.println("Iloczyn macierzy C = A*B");
        TInt.printf("%4d", c);
    }
}
```

Zadanie 28.10. Z28_10.java

Mnożenie macierzy przez liczbę (skalar) jest bardzo prostym działaniem i polega na mnożeniu każdego elementu macierzy przez tę liczbę.

```
public static double[][] product (double k, double[][] x) {
    double[][] tmp = new double[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[0].length; ++j)
            tmp[i][j] = k*x[i][j];
    return tmp;
}
```

Działanie metody pokażemy, wykonując przykładowe mnożenie.

```
public class Z28_10 {
    /* Tu wstaw kod metody product(). */
    public static void main(String[] args) {
        double[][] a = new double[2][3];
        TDouble.setRandom(a, 1.5);
        double[][] b = product(2.5, a);
        System.out.println("Macierz A");
        TDouble.printf("%5.2f", a);
        System.out.println("Macierz B = k*A");
        TDouble.printf("%6.3f", b);
    }
}
```

Zadanie 28.11. Z28_11.java

Transpozycja macierzy polega na zamianie wierszy macierzy na kolumny i kolumn na wiersze. Następuje przy tym zmiana wymiaru macierzy (nie dotyczy tzw. macierzy kwadratowych). Wystarczy zatem odczytywać macierz wierszami i zapisywać ją kolumnami (wystarczy zamiana indeksów `tmp[j][i] = x[i][j]`).

```
public static double[][] transp (double[][] x) {
    double[][] tmp = new double[x[0].length][x.length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[0].length; ++j)
            tmp[j][i] = x[i][j];
    return tmp;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

W wyniku transponowania macierzy 2×3 (dwa wiersze, trzy kolumny) powstanie macierz 3×2 (trzy wiersze, dwie kolumny).

```
public class Z28_11 {
    /* Tu wstaw kod metody transp(). */
    public static void main(String[] args) {
        double[][] a = new double[2][3];
        TDouble.setRandom(a, 100);
        double[][] b = transp(a);
        System.out.println("Macierz A");
        TDouble.printf("%6.2f", a);
        System.out.println("Macierz transponowana");
        TDouble.printf("%6.2f", b);
    }
}
```

Zadanie 28.12. Z28_12.java, TInt.java

Tworzymy tablicę (macierz) liczb zmiennoprzecinkowych, której wymiary będą identyczne z wymiarami podanego parametru (tablicy liczb całkowitych). W podwójnej pętli przepisujemy elementy z jednej tablicy do drugiej, rzutując wartość całkowitą na zmiennoprzecinkową (`tmp[i][j] = (double)x[i][j]`).

```
public class Z28_12 {
    private static double[][] toDouble (int[][] x) {
        double[][] tmp = new double[x.length][x[0].length];
        for(int i = 0; i < x.length; ++i)
            for(int j = 0; j < x[0].length; ++j)
                tmp[i][j] = (double)x[i][j];
        return tmp;
    }
    public static void main(String[] args) {
        int[][] a = new int[4][3];
        TInt.setRandom(a, 25);
        double[][] b = toDouble(a);
        System.out.println("Macierz o elementach całkowitych A");
        TInt.printf("%6d", a);
        System.out.println("Macierz o elementach rzeczywistych B");
        TDouble.printf("%6.1f", b);
    }
}
```

W przykładzie zdefiniowaliśmy metodę `toDouble()` jako prywatną. Przenosząc kod metody do klasy `TInt`, zmienimy specyfikator dostępu `private` na `public`. W przykładowym kodzie wystarczy zamienić wiersz `double[][] b = toDouble(a);` na wiersz `double[][] b = TInt.toDouble(a);` i można usunąć kod prywatnej metody `toDouble()`.

Zadanie 28.13. Z28_13.java, TDouble.java

Metoda `valueOf()` w klasie `TDouble` zamienia tablicę typu `int[][]` na tablicę typu `double[][]`, więc jej kod jest identyczny z metodą `TInt.toDouble()` (rozwiązanie zadania 28.12).

```
public static double[][] valueOf (int[][] x) {
    double[][] tmp = new double[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(int j = 0; j < x[0].length; ++j)
                tmp[i][j] = (double)x[i][j];
        return tmp;
    }
```

Program demonstrujący działanie metody jest również podobny do programu z rozwiązania zadania 28.12. Różnica polega na zastąpieniu metody `TInt.toDouble()` metodą `TDouble.valueOf()`.

Zadanie 28.14. Z28_14.java, TDouble.java

Wzorując się na rozwiązaniu zadania 28.12, utworzymy metodę `toInt()` i dołączymy ją do klasy `TDouble`.

```
public static int[][] toInt (double[][] x) {
    int[][] tmp = new int[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[0].length; ++j)
            tmp[i][j] = (int)x[i][j];
    return tmp;
}
```

Przykład działania metody możemy zbudować podobnie jak w rozwiązaniu zadania 28.12.

Zadanie 28.15. Z28_15.java, TInt.java

Wzorując się na rozwiązaniu zadania 28.13 (metoda `valueOf()` umieszczona w klasie `TDouble`), zbudujemy metodę `valueOf()` w klasie `TInt`.

```
public static int[][] valueOf (double[][] x) {
    int[][] tmp = new double[x.length][x[0].length];
    for(int i = 0; i < x.length; ++i)
        for(int j = 0; j < x[0].length; ++j)
            tmp[i][j] = (int)x[i][j];
    return tmp;
}
```

Przykład działania metody możemy zbudować, wzorując się na rozwiązaniu jednego z powyższych zadań.

Zadanie 28.16. Z28_16.java

Ślad macierzy (pojęcie z algebry liniowej) jest sumą elementów na głównej przekątnej macierzy kwadratowej.

```
public class Z28_16 {
    private static double trace (double[][] x) {
        if (x.length != x[0].length)
            throw new ArithmeticException("Macierz nie jest kwadratowa");
        double tr = 0.0;
        for(int i = 0; i < x.length; ++i)
            tr += x[i][i];
        return tr;
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        double[][] a = new double[4][4];
        TDouble.setRandom(a, 100);
        System.out.println("Macierz A");
        TDouble.printf("%8.3f", a);
        System.out.printf("Ślad macierzy tr(A) = %.3f\n", trace(a));
    }
}
```

Zadanie 28.17. Z28_17.java

Metoda `getI()` zwraca macierz jednostkową rzędu *n*, gdzie *n* jest parametrem wywołania metody. Macierzą jednostkową lub identycznościową nazywamy macierz kwadratową, która na przekątnej głównej ma wartości *1* (dla $i = j$), a pozostałe wartości są równe *0* (dla $i \neq j$). Ponieważ po utworzeniu (`double[][] tmp = new double[n][n];`) tablica jest wypełniona zerami, to wystarczy wstawić na głównej przekątnej wartość *1* (`for(int i = 0; i < n; ++i) tmp[i][i] = 1.0;`).

```
public class Z28_17 {
    private static double[][] getI (int n) {
        if (n < 1)
            throw new ArithmeticException("Niewłaściwy wymiar macierzy");
        double[][] tmp = new double[n][n];
        for(int i = 0; i < n; ++i)
            tmp[i][i] = 1.0;
        return tmp;
    }
    public static void main(String[] args) {
        double[][] a = getI(5);
        System.out.println("Macierz jednostkowa 5 stopnia");
        TDouble.printf("%5.1f", a);
    }
}
```

Program wyświetla macierz jednostkową *5.* rzędu. Czytelnik może do programu dołączyć metodę mnożącą macierze i pokazać (na przykładach) równości $I \cdot A = A$ i $B \cdot I = B$.

Zadanie 28.18. Z28_18.java

Ponieważ macierz `x`, przekazana jako parametr wywołania metody `setI()`, może zawierać jakieś wartości, musimy na głównej przekątnej wstawić liczby *1* (dla $i = j$), a inne wartości pozostawić równe *0* (dla $i \neq j$).

```
public class Z28_18 {
    private static void setI (double[][] x) {
        if (x.length != x[0].length)
            throw new ArithmeticException("Niewłaściwy wymiar macierzy");
        for(int i = 0; i < x.length; ++i)
            for(int j = 0; j < x[0].length; ++j)
                if (i == j)
                    x[i][j] = 1.0;
                else
                    x[i][j] = 0.0;
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        double[][] a = new double[4][4];
        setI(a);
        System.out.println("Macierz jednostkowa");
        TDouble.printf("%4.1f", a);
    }
}
```

Zadanie 28.19. Z28_19.java

Wyznacznik obliczymy z *rozwinięcia Laplace'a* względem pierwszego wiersza (wiersza o indeksie *0*). Wzór wynikający z tego rozwinięcia jest rekurencyjny[1] i obliczenie wyznacznika macierzy *n*-tego rzędu sprowadzi się do obliczenia *n* wyznaczników rzędu *n–1*.

```
private static double det(double[][] x) {
    if (x.length != x[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int st = x.length;
    double d = 0;
    if (st == 1)
        d = x[0][0];
    else {
        int zn = 1;
        double[][] n = new double[st-1][st-1];
        for(int i = 0; i < st; ++i) {
            for(int j = 0; j < st-1; ++j) {
                for(int k = 0; k < st-1; ++k) {
                    if (k < i)  n[j][k] = x[j+1][k];
                        else  n[j][k] = x[j+1][k+1];
                }
            }
            d += zn*x[0][i]*det(n);
            zn = -zn;
        }
    }
    return d;
}


public class Z28_19 {
    /* Tu wstaw kod metody det(). */
    public static void main(String[] args) {
        double[][] a = new double[6][6];
        TDouble.setRandom(a, 10.0);
        System.out.println("Macierz A");
        TDouble.printf("%6.2f", a);
        System.out.printf("Wyznacznik det A = %6.2f\n", det(a));
    }
}
```

[1] Rekurencja polega na odwoływaniu się metody (funkcji, definicji) do samej siebie. W rozdziale 29. Czytelnik znajdzie więcej zadań, których rozwiązania zawierają metody rekurencyjne.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.20. Z28_20.java

Macierz trójkątna górna to macierz kwadratowa, w której wszystkie elementy znajdujące się poniżej głównej przekątnej mają wartość równą *0*. Możemy przyjąć jedno z dwóch rozwiązań: przekształcenia wykonujemy na macierzy przekazanej jako parametr wywołania metody, przekształcenia wykonujemy na kopii parametru wywołania i zwracamy przekształconą kopię jako wynik działania metody (przyjmiemy ten drugi wariant).

Jeśli podana macierz nie jest kwadratowa, to zgłaszany jest wyjątek. Po utworzeniu kopii (`tmp`) przekształcanej macierzy w pętli (`for(int i = 0; i < n-1; ++i)`) pobieramy elementy z głównej przekątnej (`tmp[i][i]`) — od pierwszego do przedostatniego. Dla wszystkich wierszy o indeksie `j > i` odejmujemy od wiersza o indeksie `j` wiersz o indeksie `i` pomnożony przez współczynnik `p = tmp[j][i]/tmp[i][i]`. Operację możemy wykonać, gdy element `tmp[i][i]` jest różny od zera (elementy `tmp[j][i]` przyjmują wartość *0*).

```
if (tmp[i][i] != 0.0) {
    double p = tmp[j][i]/tmp[i][i];
    for(int k = 0; k < n; ++k)
        tmp[j][k] -= tmp[i][k]*p;
}
```

Ta operacja nie zmienia wyznacznika macierzy. Natomiast elementy macierzy znajdujące się poniżej elementu `tmp[i][i]` w kolumnie `i` przyjmują wartość *0*.

Jeśli element `t[i][i]` jest zerem, to szukamy w wierszach poniżej najbliższego wiersza `k` takiego, że element `tmp[k][i]` jest różny od zera. Przestawiamy te wiersze i w ten sposób na przekątnej (w `tmp[i][i]`) znajdzie się element różny od zera. Przestawianie wierszy (lub kolumn) macierzy powoduje zmianę znaku wyznacznika. Aby temu zapobiec, mnożymy przestawiony wiersz przez *–1* (realizuje to zmiana znaku `temp` w operacji przestawiania elementów `tmp[k][m] = -temp`). Po przestawieniu wierszy kontynuujemy działanie pętli ze zmienną `j` (odejmujemy od wiersza o indeksie `j` wiersz o indeksie `i` pomnożony przez współczynnik `p`). Gdyby się okazało, że dla `k > i` nie ma wiersza z elementem `tmp[k][i]` różnym od zera, to wartość *0* pozostanie na głównej przekątnej.

```
private static double[][] upperTriangular(double[][] x) {
    if (x.length != x[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    /* Kopia macierzy */
    int n = x.length;
    double[][] tmp = new double[n][n];
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            tmp[i][j] = x[i][j];
    /* Tworzenie macierzy trójkątnej górnej */
    for(int i = 0; i < n-1; ++i)
        for(int j = i+1; j < n; ++j) {
            if (tmp[i][i] != 0.0) {
                double p = tmp[j][i]/tmp[i][i];
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                for(int k = 0; k < n; ++k)
                    tmp[j][k] -= tmp[i][k]*p;
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp[i][m];
                            tmp[i][m] = tmp[k][m];
                            tmp[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
    return tmp;
}
```

Jak już wspomnieliśmy, przedstawiony algorytm zamiany macierzy kwadratowej na macierz trójkątną górną nie zmienia wartości wyznacznika macierzy. Wyznacznik macierzy trójkątnej jest natomiast równy iloczynowi elementów należących do głównej przekątnej, zatem przedstawiona metoda może być wykorzystana do obliczenia wyznacznika.

```
public class Z28_20 {
    /* Tu wstaw kod metody upperTriangular. */
    public static void main(String[] args) {
        double[][] a = new double[6][6];
        TDouble.setRandom(a, 10.0);
        System.out.println("Macierz A");
        TDouble.printf("%8.2f", a);
        double[][] b = upperTriangular(a);
        System.out.println("Macierz trójkątna górna B");
        TDouble.printf("%8.2f", b);
        /* Wyznacznik macierzy trójkątnej */
        double det = 1.0;
        for(int i = 0; i < b.length; ++i)
            det *= b[i][i];
        System.out.printf("det A = det B = %.2f\n", det);
    }
}
```

Zadanie 28.21. Z28_21.java

Macierz trójkątna dolna to macierz kwadratowa, w której wszystkie elementy znajdujące się ponad główną przekątną mają wartość równą *0*.

Sposób tworzenia *macierzy trójkątnej dolnej* jest podobny do sposobu *tworzenia macierzy trójkątnej górnej* (zob. rozwiązanie zadania 28.20). Różnica polega na kolejności pobierania elementów z głównej przekątnej macierzy — elementy pobieramy od ostatniego do drugiego (`for(int i = n-1; i > 0; --i)`). Wykonujemy odejmowanie wierszy (`for(int j = i-1; j >= 0; --j)`) i poszukiwanie wiersza z niezerowym elementem (`for(int k = i-1; k >= 0; --k)`) „od dołu do góry”.

Ebookpoint.pl kopia dla: pawher@o2.pl

```
private static double[][] lowerTriangular(double[][] x) {
    if (x.length != x[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = x.length;
    double[][] tmp = new double[n][n];
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            tmp[i][j] = x[i][j];
    for(int i = n-1; i > 0; --i)
        for(int j = i-1; j >= 0; --j) {
            if (tmp[i][i] != 0.0) {
                double p = tmp[j][i]/tmp[i][i];
                for(int k = 0; k < n; ++k)
                    tmp[j][k] -= tmp[i][k]*p;
            } else {
                for(int k = i-1; k >= 0; --k) {
                    if (tmp[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp[i][m];
                            tmp[i][m] = tmp[k][m];
                            tmp[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
    return tmp;
}
```

Przedstawiony algorytm zamiany macierzy kwadratowej na *macierz trójkątną dolną* nie zmienia wartości wyznacznika macierzy. Wyznacznik macierzy trójkątnej jest równy iloczynowi elementów należących do głównej przekątnej.

Zadanie 28.22. Z28_22.java

Macierz diagonalna to macierz kwadratowa, której wszystkie współczynniki leżące poza główną przekątną są zerowe (jest to *macierz trójkątna górna* i zarazem *macierz trójkątna dolna*). Możemy zatem zastosować kolejno metody `upperTriangular()` (zadanie 28.20) i `lowerTriangular()` (zadanie 28.21) — otrzymamy *macierz diagonalną*.

W metodzie `upperTriangular()` możemy dokonać kilku poprawek i uzyskamy metodę wyznaczającą *macierz diagonalną*:

- zmiana nazwy metody `upperTriangular` na `diagonal`,
- zastąpienie pętli `for(int j = i+1; j < n; ++j)` pętlą `for(int j = 0; j < n; ++j)`, co spowoduje odejmowanie od wszystkich wierszy macierzy wiersza o indeksie `i` pomnożonego przez współczynnik `p = tmp[j][i]/tmp[i][i]`,
- w powyższym odejmowaniu należy pominąć odejmowanie wiersza od siebie samego ($j = i$), więc operację wykonujemy warunkowo:

```
if (j != i) {
    double p = tmp[j][i]/tmp[i][i];
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(int k = 0; k < n; ++k)
                tmp[j][k] -= tmp[i][k]*p;
        }
```

Po tych zmianach otrzymamy następujący kod metody:

```
private static double[][] diagonal(double[][] x) {
    if (x.length != x[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    /* Kopia macierzy */
    int n = x.length;
    double[][] tmp = new double[n][n];
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            tmp[i][j] = x[i][j];
    /* Wyznaczenie macierzy diagonalnej */
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j) {
            if (tmp[i][i] != 0.0) {
                if (j != i) {
                    double p = tmp[j][i]/tmp[i][i];
                    for(int k = 0; k < n; ++k)
                        tmp[j][k] -= tmp[i][k]*p;
                }
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double p = tmp[i][m];
                            tmp[i][m] = tmp[k][m];
                            tmp[k][m] = -p;
                        }
                        break;
                    }
                }
            }
        }
    return tmp;
}
```

Program demonstrujący działanie może być podobny jak w zadaniu 28.20 lub 28.21.

Zadanie 28.23. Z28_23.java

Do wyznaczenia macierzy odwrotnej zastosujemy następujący algorytm:

- Bierzemy daną macierz kwadratową (A) i macierz jednostkową (I) tego samego rzędu.
- Stosując te same przekształcenia dla obu macierzy jednocześnie (mnożenie wierszy przez liczbę różną od zera, dodawanie do wiersza macierzy kombinacji liniowej innych wierszy), doprowadzamy macierz A do postaci macierzy diagonalnej z wartościami *1* na głównej przekątnej. Macierz powstała w wyniku przekształceń macierzy I jest macierzą odwrotną do macierzy A (A^{-1}).

Ebookpoint.pl kopia dla: pawher@o2.pl

Jeśli w wyniku przekształceń macierzy *A* otrzymamy macierz o elemencie *0* na głównej przekątnej, to macierz *A* jest *macierzą osobliwą* (*zdegenerowaną*); nie istnieją wyznacznik *det A = 0* i macierz odwrotna.

W kodzie metody `diagonal()` (z rozwiązania zadania 28.22) zmieniono nazwę na `inverse` i dodano fragmenty działań (wykonywanych równolegle) na macierzy jednostkowej *I* (na listingu zmiany i uzupełnienia w kodzie zaznaczono pogrubieniem). Po sprowadzeniu macierzy `tmp` do postaci *diagonalnej* sprawdzamy, czy nie jest to *macierz osobliwa* (*0* na głównej przekątnej, rzucamy wyjątek). W trakcie tej operacji odpowiednie wiersze obu macierzy dzielimy przez `tmp[i][i]`, co doprowadza macierz `tmp` do macierzy jednostkowej, a macierz `I` do macierzy odwrotnej do wyjściowej postaci macierzy `tmp`. Zwracamy wynik znajdujący się w macierzy *I*.

```
private static double[][] inverse(double[][] x) {
    if (x.length != x[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = x.length;

    double[][] tmp = new double[n][n];
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
            tmp[i][j] = x[i][j];

    double[][] I = new double[n][n];
    for(int i = 0; i < n; ++i)
        I[i][i] = 1.0;

    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j) {
            if (tmp[i][i] != 0.0) {
                if (j != i) {
                    double p = tmp[j][i]/tmp[i][i];
                    for(int k = 0; k < n; ++k) {
                        tmp[j][k] -= tmp[i][k]*p;
                        I[j][k] -= I[i][k]*p;
                    }
                }
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp[i][m];
                            tmp[i][m] = tmp[k][m];
                            tmp[k][m] = -temp;
                            temp = I[i][m];
                            I[i][m] = I[k][m];
                            I[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
       }
    for(int i = 0; i < n; ++i)
        if (tmp[i][i] != 0.0)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                for(int j = 0; j < n; ++j)
                    I[i][j] /= tmp[i][i];
            else
                throw new ArithmeticException("Macierz odwrotna nie
                    istnieje");
        return I;
    }
```

Spróbujemy uprościć uzyskaną metodę. Oto istotny fragment kodu:

```
for(int i = 0; i < n; ++i) {
    for(int j = 0; j < n; ++j) {
        if (tmp[i][i] != 0.0) {
            if (j != i) {
                double p = tmp[j][i]/tmp[i][i];
                for(int k = 0; k < n; ++k) {
                    tmp[j][k] -= tmp[i][k]*p;
                    I[j][k] -= I[i][k]*p;
                }
            }
        } else {
            for(int k = i+1; k < n; ++k) {
                if (tmp[k][i] != 0.0) {
                    for(int m = 0; m < n; ++m) {
                        double temp = tmp[i][m];
                        tmp[i][m] = tmp[k][m];
                        tmp[k][m] = -temp;
                        temp = I[i][m];
                        I[i][m] = I[k][m];
                        I[k][m] = -temp;
                    }
                    break;
                }
            }
            throw new ArithmeticException("Macierz odwrotna nie
                istnieje");// 0 na przekątnej macierzy diagonalnej
        }
    }
    double q = tmp[i][i];
    for(int k = 0; k < n; ++k) {
        I[i][k] /= q;
        tmp[i][k] /= q;
    }
}
return I;
```

Analizę zmian pozostawiamy Czytelnikowi.

Demonstrując działanie metody, sprawdzimy, czy uzyskana macierz jest odwrotna. W tym celu pomnożymy macierz wyjściową przez macierz odwrotną. W rozwiązaniu zadania 28.9 zbudowaliśmy metodę `product()`, obliczającą iloczyn macierzy o elementach całkowitych. Na podstawie tej metody zbudujemy metodę mnożącą macierze o elementach rzeczywistych:

```
public static double[][] product (double[][] x, double[][] y) {
    if (x[0].length != y.length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        double[][] tmp = new double[x.length][y[0].length];
        for(int i = 0; i < x.length; ++i)
            for(int j = 0; j < y[0].length; ++j)
                for(int k = 0; k < y.length; ++k)
                    tmp[i][j] += x[i][k]*y[k][j];
        return tmp;
    }
```

Używając metody `product()`, możemy sprawdzić na przykładach działanie metody `inverse()`:

```
    public class Z28_23a {
        /* Tu wstaw kod metody inverse(). */
        /* Tu wstaw kod metody product(). */
        public static void main(String[] args) {
            double[][] a = new double[4][4];
            TDouble.setRandom(a, 10.0);
            System.out.println("Macierz A");
            TDouble.printf("%8.2f", a);
            double[][] b = inverse(a);
            System.out.println("Macierz odwrotna B");
            TDouble.printf("%8.2f", b);
            System.out.println("Sprawdzenie A·B");
            TDouble.printf("%8.2f", product(a, b));
            System.out.println("Sprawdzenie B·A");
            TDouble.printf("%8.2f", product(b, a));
            double[][] c = {{1, 1, 1},{1, 1, 1}, {2, 2, 2}};
            System.out.println("Macierz C");
            TDouble.printf("%8.2f", c);
            System.out.println("Macierz odwrotna D");
            double[][] d = inverse(c);
            TDouble.printf("%8.2f", d);;
        }
    }
```

Zadanie 28.24. Z28_24.java

Rozwiążemy układ *n* równań liniowych o *n* niewiadomych:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Na początku programu wprowadzimy liczbę równań układu (*n*). Współczynniki układu równań wprowadzimy do tablicy o *n* wierszach i *n+1* kolumnach. W pierwszych *n* kolumnach umieścimy macierz podstawową układu współrzędnych, a w ostatniej kolumnie umieścimy wyrazy wolne. Do wprowadzania danych użyjemy metody `input()` (z rozwiązania zadania 28.3) z niewielkimi poprawkami:

```
    private static void input(double[][] x) {
        int n = x.length;
        for(int i = 0; i < n; ++i) {
            System.out.println("Podaj współczynniki równania");
            for(int j = 0; j < n; ++j) {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            System.out.printf("a[%d][%d] = ", i+1, j+1);
            x[i][j] = cin.nextDouble();
        }
        System.out.printf("b[%d] = ", i+1);
        x[i][n] = cin.nextDouble();
    }
}
```

Do obliczania wyznaczników użyjemy metody rekurencyjnej `det()` z rozwiązania zadania 28.19. Ponieważ tablica przechowująca współczynniki układu równań nie jest tablicą kwadratową (liczba kolumn jest o jeden większa od liczby wierszy), to z metody `det()` usuwamy kontrolę wymiaru. Wyznacznik będzie liczony z pierwszych *n* kolumn macierzy, a *n+1* kolumna (zawierająca wyrazy wolne) zostanie pominięta. W ten sposób obliczymy główny wyznacznik układu współrzędnych (`w`).

```
private static double det (double[][] x) {
    int st = x.length;
    double d = 0;
    if (st == 1)
        d = x[0][0];
    else {
        int zn = 1;
        double[][] n = new double[st-1][st-1];
        for (int i = 0; i < st; ++i) {
            for (int j = 0; j < st-1; ++j) {
                for (int k = 0; k < st-1; ++k) {
                    if (k < i)  n[j][k] = x[j+1][k];
                        else  n[j][k] = x[j+1][k+1];
                }
            }
            d += zn*x[0][i]*det(n);
            zn = -zn;
         }
    }
    return d;
}
```

Pozostałe wyznaczniki obliczymy (w pętli) w ten sam sposób, ale przed liczeniem wyznacznika w miejsce *i*-tej kolumny podstawimy (ostatnią) kolumnę wyrazów wolnych. Do tego celu utworzymy metodę `swapLast()`:

```
private static void swapLast(double[][] x, int col) {
    int n = x.length;
    for(int row = 0; row < n; ++row) {
        double tmp = x[row][col];
        x[row][col] = x[row][n];
        x[row][n] = tmp;
    }
}
```

Po obliczeniu wyznacznika ponownie wywołamy metodę `swapLast()`, przywracając pierwotną postać macierzy.

Po obliczeniu wyznaczników możemy podać rozwiązanie układu równań. Możliwe są trzy przypadki:

Ebookpoint.pl kopia dla: pawher@o2.pl

1. Wyznacznik główny (w) jest różny od zera, więc *układ równań jest oznaczony* $x_i = \frac{w_i}{w}$ dla $i = 1, 2, \ldots, n$.
2. Wyznacznik główny jest równy *0* i wszystkie pozostałe wyznaczniki są równe 0 (suma s bezwzględnych wartości tych wyznaczników jest równa zero), więc *układ równań jest nieoznaczony*.
3. Wyznacznik główny jest równy *0* i co najmniej jeden z pozostałych wyznaczników nie jest równy 0 (suma s bezwzględnych wartości tych wyznaczników nie jest zerem), zatem *układ równań jest sprzeczny*.

```
import java.util.Scanner;
public class Z28_24 {
    private static Scanner cin = new Scanner(System.in);
    /* Tu wstaw kod metody det(). */
    /* Tu wstaw kod metody swapLast(). */
    /* Tu wstaw kod metody input(). */

    public static void main(String[] args) {
        /* Liczba równań */
        int n;
        do {
            System.out.print("Podaj liczbę równań układu (2..9), n = ");
            n = cin.nextInt();
        } while (n < 2 || n > 9);
        /* Wprowadzenie współczynników układu równań */
        double[][] ur = new double[n][n+1];
        input(ur);
        /* Obliczanie wyznaczników */
        double w = det(ur);
        System.out.printf("Wyznacznik główny W = %f\n", w);
        double[] wx = new double[n];
        for(int i = 0; i < n; ++i) {
            swapLast(ur, i);
            wx[i] = det(ur);
            swapLast(ur, i);
            System.out.printf("Wyznacznik Wx[%d] = %f\n", i+1, wx[i]);
        }
        if (w != 0) {
            System.out.println("Układ równań jest oznaczony");
            for(int i = 0; i < n; ++i)
               System.out.printf("x[%d] = %f\n", i+1, wx[i]/w);
        } else {
            double s = 0.0;
            for(int i = 0; i < n; ++i)
               s += Math.abs(wx[i]);
            if (s == 0.0)
                System.out.println("Układ równań jest nieoznaczony");
            else
                System.out.println("Układ równań jest sprzeczny");
        }
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 28.25. Z28_25.java

Na początku programu wprowadzimy liczbę równań układu (n). Współczynniki układu równań

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

wprowadzimy do dwóch tablic (macierzy) — kwadratowej macierzy podstawowej układu o wymiarze $n \times n$ i prostokątnej o wymiarze $n \times 1$ (kolumna wyrazów wolnych). Do wprowadzania danych użyjemy metody `input()` (z rozwiązania zadania 28.24) z poprawkami uwzględniającymi przechowywanie danych w dwóch macierzach:

```
private static void input(double[][] x, double[][] y) {
    int n = x.length;
    for(int i = 0; i < n; ++i) {
        System.out.println("Podaj współczynniki równania");
        for(int j = 0; j < n; ++j) {
            System.out.printf("a[%d][%d] = ", i+1, j+1);
            x[i][j] = cin.nextDouble();
        }
        System.out.printf("b[%d] = ", i+1);
        y[i][0] = cin.nextDouble();
    }
}
```

Z rozwiązania zadania 28.23 weźmiemy metodę `inverse()` obliczającą macierz odwrotną oraz metodę `product()` mnożącą dwie macierze. Jeśli macierz odwrotna istnieje, to układ jest oznaczony, w przeciwnym wypadku (metoda `inverse()` rzuci wyjątek) mamy do czynienia z układem sprzecznym lub nieoznaczonym.

```
import java.util.Scanner;
public class Z28_25 {
    private static Scanner cin = new Scanner(System.in);

    /* Tu wstaw kod metody input(). */
    /* Tu wstaw kod metody inverse() - zadanie 28.23. */
    /* Tu wstaw kod metody product() - zadanie 28.23 */

    public static void main(String[] args) {
        /* Liczba równań */
        int n;
        do {
            System.out.print("Podaj liczbę równań układu, n = ");
            n = cin.nextInt();
        } while (n < 2);
        /* Wprowadzenie współczynników układu równań */
        double[][] a = new double[n][n];
        double[][] b = new double[n][1];
        input(a, b);
        /* Obliczanie macierzy odwrotnej i rozwiązanie układu */
        try {
            double[][] x = product(inverse(a), b);
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            System.out.println("Układ równań jest oznaczony");
            for(int i = 0; i < n; ++i)
                System.out.printf("x[%d] = %f\n", i+1, x[i][0]);
        } catch (ArithmeticException e) {
            System.out.println("Układ sprzeczny lub nieoznaczony");
        }
    }
}
```

Zadanie 28.26. Z28_26.java

Na początku programu wprowadzimy liczbę równań układu (*n*). Współczynniki układu równań wprowadzimy do tablicy o *n* wierszach i *n+1* kolumnach (tak jak w rozwiązaniu zadania 28.24). W pierwszych *n* kolumnach umieścimy macierz podstawową układu współrzędnych, a w ostatniej kolumnie — wyrazy wolne. Do wprowadzania danych użyjemy metody `input()` z rozwiązania zadania 28.4. Następnie będziemy przekształcali macierz zawierającą współczynniki równań układu w taki sposób, aby macierz podstawowa stała się macierzą jednostkową. W ostatniej kolumnie otrzymamy rozwiązanie układu.

Jeśli na głównej przekątnej macierzy podstawowej pojawi się wartość *0*, to układ jest sprzeczny lub nieoznaczony (możemy to rozstrzygnąć, porównując zerowe wartości z przekątnej macierzy podstawowej z wartościami z kolumny wyrazów wolnych).

Metoda `solve()`, rozwiązująca układ, jest modyfikacją metody `inverse()`, obliczającej macierz odwrotną (zob. rozwiązanie zadania 28.23). Metoda `solve()` nie sprawdza wymiaru macierzy (powinien być równy $n \times n+1$). Najpierw macierz jest przekształcana tak, aby macierz podstawowa stała się macierzą diagonalną; te same operacje są wykonywane na kolumnie wyrazów wolnych. W tym momencie otrzymujemy *n* równań o postaci $ax = b$ (gdzie *a* jest elementem przekątnej macierzy podstawowej, *b* odpowiadającym mu wyrazem wolnym). W ostatniej pętli rozwiązujemy te równania ($x = b/a$) lub stwierdzamy, że $a = 0$ i równanie jest sprzeczne (*b* różne od *0*) lub nieoznaczone ($b = 0$) — zwracamy wyjątek, nie informując o powodach. Czytelnik może ten fragment kodu rozwinąć. Jeśli co najmniej jedno równanie jest sprzeczne, to cały układ też jest sprzeczny.

```
private static void solve(double[][] x) {
    int n = x.length;
    /* Sprowadzamy macierz podstawową do postaci diagonalnej. */
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j) {
            if (x[i][i] != 0.0) {
                if (j != i) {
                    double p = x[j][i]/x[i][i];
                    for(int k = 0; k < n+1; ++k)
                     x[j][k] -= x[i][k]*p;
                 }
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (x[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = x[i][m];
                            x[i][m] = x[k][m];
                            x[k][m] = -temp;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
                        }
                        break;
                    }
                }
            }
        }
    /* Wyznaczamy rozwiązanie układu. */
    for(int i = 0; i < n; ++i)
        if (x[i][i] != 0.0)
            for(int j = 0; j < n; ++j) {
                x[i][n] /= x[i][i];
                x[i][i] = 1.0;
            }
        else if (x[i][i] == x[i][n])
            throw new ArithmeticException("Układ sprzeczny lub nieoznaczony");
}
```

W przykładowym rozwiązaniu nie ograniczamy liczby równań. W odróżnieniu od metody wyznaczników budującej wiele kopii macierzy metoda eliminacji przekształca macierz układu równań w miejscu, gdzie ją utworzono (macierz ulega zmianie). Jeśli będzie nam potrzebna początkowa postać układu równań, to musimy przed wywołaniem metody `solve()` skopiować dane do innej tablicy.

```
import java.util.Scanner;
public class Z28_26 {
    private static Scanner cin = new Scanner(System.in);

    /* Tu wstaw kod metody input(). */
    /* Tu wstaw kod metody solve(). */

    public static void main(String[] args) {
        /* Liczba równań */
        int n;
        do {
            System.out.print("Podaj liczbę równań układu, n = ");
            n = cin.nextInt();
        } while (n < 2);
        /* Wprowadzenie współczynników układu równań */
        double[][] ur = new double[n][n+1];
        input(ur);
        /* Rozwiązanie układu */
        try {
            solve(ur);
            for(int i = 0; i < n; ++i)
                System.out.printf("x[%d] = %f\n", i+1, ur[i][n]);
        } catch (ArithmeticException e) {
            System.out.println("Układ sprzeczny lub nieoznaczony");
        }
    }
}
```

Zadanie 28.27. Matrix.java, Z28_27.java

W klasie `Matrix` korzystamy z metod pochodzących z dwóch klas, `Scanner` i `Random`, z pakietu `java.util`. W tym celu tworzymy prywatne i statyczne obiekty `cin` (ang. *console input*) i `rnd` (ang. *random*). Ma to na celu ułatwienie pracy z klasą podczas jej

Ebookpoint.pl kopia dla: pawher@o2.pl

tworzenia i testowania jej możliwości (`random()` — wypełnianie macierzy wylosowanymi wartościami) oraz w czasie wprowadzania danych (metoda `input()`) i wyświetlania wyników (metoda `printf()`).

Obiekt posiada prywatne pole `M`[2] będące referencją do dwuwymiarowej tablicy liczb typu `double`, zawierającej elementy macierzy. Do dyspozycji mamy trzy konstruktory: tworzący macierz na podstawie istniejącej tablicy dwuwymiarowej, tworzący macierz o podanych wymiarach (wypełnioną zerami) i konstruktor kopiujący.

Oprócz wspomnianych metod (`random()`, `input()` i `printf()`) w klasie `Matrix` zdefiniowano metody umożliwiające dostęp do prywatnych danych (elementów tablicy `M`, czyli elementów macierzy). Metoda `getAt()` pozwala odczytać wartość elementu z określonego wiersza i kolumny, a metoda `setAt()` nadaje określonemu elementowi podaną wartość. Wymiary macierzy możemy poznać, korzystając z metod `getRows()` i `getColumns()`. Dzięki tym metodom zewnętrzne metody mogą przekształcać obiekty reprezentujące macierze (o ile takich operacji nie oferują metody klasy `Matrix`).

```
import java.util.*;
public class Matrix {
    private static Scanner cin = new Scanner(System.in);
    private static Random  rnd = new Random();

    private double[][] M;

    public Matrix(double[][] x) {
        int r = x.length;     // liczba wierszy (rows)
        int c = x[0].length; // liczba kolumn (columns)
        M = new double[r][c];
        for(int i = 0; i < r; ++i)
            for(int j = 0; j < c; ++j)
                M[i][j] = x[i][j];
    }

    public Matrix(int r, int c) {
        /* r liczba wierszy (rows), c liczba kolumn (columns) */
        M = new double[r][c];
    }

    public Matrix(Matrix x) {
        int r = x.M.length;
        int c = x.M[0].length;
        M = new double[r][c];
        for(int i = 0; i < r; ++i)
            for(int j = 0; j < c; ++j)
                M[i][j] = x.M[i][j];
    }

    public Matrix random() {
        for(int i = 0; i < M.length; ++i)
```

[2] Gdy programujemy w Javie, jesteśmy przyzwyczajeni do zapisywania identyfikatorów stałych wielkimi literami. W tym przypadku `M` nie oznacza stałej, lecz jest identyfikatorem tablicy, która odpowiada macierzy reprezentowanej przez obiekt. W kilku następnych zadaniach będziemy oznaczali macierze wielkimi literami (tak jak przyjmuje się w podręcznikach do matematyki).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(int j = 0; j < M[0].length; ++j)
                M[i][j] = rnd.nextDouble();
        return this;
    }

    public Matrix input() {
        for(int i = 0; i < M.length; ++i)
            for(int j = 0; j < M[i].length; ++j) {
                System.out.printf("M[%d][%d] = ", i, j);
                M[i][j] = cin.nextDouble();
            }
        return this;
    }

    public Matrix printf(String format) {
        for(int i = 0; i < M.length; ++i) {
            for(int j = 0; j < M[0].length; ++j)
                System.out.printf(format, M[i][j]);
            System.out.println();
        }
        return this;
    }

    public double getAt(int i, int j) {
        return M[i][j];
    }

    public void setAt(int i, int j, double x) {
        M[i][j] = x;
    }

    public int getRows() {
        return M.length;
    }

    public int getColumns() {
        return M[0].length;
    }
}
```

W przykładzie pokazujemy możliwości tak zdefiniowanej klasy `Matrix`.

```
class Z28_27 {
    public static void main(String[] args)  {
        double[][] a = {{0, 3, -2, 2},{2, 1, -3, -1},{2, 1, -3, 4} };

        Matrix A = new Matrix(a);
        System.out.println("Macierz A");
        A.printf("%8.2f");

        Matrix B = new Matrix(3, 4);
        System.out.println("Wypełnienie macierzy B wartościami
            losowymi");
        B.random().printf("%8.2f");

        Matrix C = new Matrix(a);
        System.out.println("Konstruktor kopiujący - C jest kopią A");
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        C.random().printf("%8.2f");
        System.out.println("Liczba wierszy: "+C.getRows());
        System.out.println("Liczba kolumn: "+C.getColumns());
        System.out.println("Element C[0][1]: "+C.getAt(0, 1));
        System.out.println("Podstawiam C[0][0] = -3.5");
        C.setAt(0, 1, -3.5);
        System.out.println("Element C[0][0]: "+C.getAt(0, 1));

        Matrix D = new Matrix(2, 2);
        System.out.println("Wprowadzanie danych z konsoli do
            macierzy D");
        D.input();
        System.out.println("Macierz D");
        D.printf("%10.4f");
    }
}
```

Zadanie 28.28. Matrix.java, Z28_28.java

Do klasy `Matrix` dodajemy metody `add()` (dodawanie), `sub()` (odejmowanie) i `mult()` (mnożenie) w dwóch wariantach — mnożenie macierzy przez macierz i mnożenie macierzy przez skalar.

```
public Matrix add(Matrix x) {
    if (M.length != x.M.length || M[0].length != x.M[0].length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < M.length; ++i)
        for(int j = 0; j < M[0].length; ++j)
            tmp.M[i][j] = M[i][j]+x.M[i][j];
    return tmp;
}

public Matrix sub(Matrix x) {
    if (M.length != x.M.length || M[0].length != x.M[0].length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < M.length; ++i)
        for(int j = 0; j < M[0].length; ++j)
            tmp.M[i][j] = M[i][j]-x.M[i][j];
    return tmp;
}

public Matrix mult(Matrix x) {
    if (M[0].length != x.M.length)
        throw new ArithmeticException("Niezgodne wymiary macierzy");
    Matrix tmp = new Matrix(M.length, x.M[0].length);
    for(int i = 0; i < M.length; ++i)
        for(int j = 0; j < x.M[0].length; ++j)
            for(int k = 0; k < x.M.length; ++k)
                tmp.M[i][j] += M[i][k]*x.M[k][j];
    return tmp;
}

public Matrix mult(double x) {
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < M.length; ++i)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            for(int j = 0; j < M[0].length; ++j)
                tmp.M[i][j] *= x;
        return tmp;
    }
```

Na macierzach wypełnionych wylosowanymi wartościami pokazujemy wykonywanie zdefiniowanych w klasie `Matrix` działań na macierzach.

```
class Z28_28 {
    public static void main(String[] args)  {
        System.out.println("Wypełnienie macierzy A wartościami
            losowymi");
        Matrix A = new Matrix(3, 4).random().printf("%8.2f");
        System.out.println("Wypełnienie macierzy B wartościami
            losowymi");
        Matrix B = new Matrix(3, 4).random().printf("%8.2f");

        System.out.println("Suma macierzy, A+B = ");
        A.add(B).printf("%8.2f");
        System.out.println("Suma macierzy, B+A = ");
        B.add(A).printf("%8.2f");

        System.out.println("Różnica macierzy, A-B = ");
        A.sub(B).printf("%8.2f");
        System.out.println("Różnica macierzy, B-A = ");
        B.sub(A).printf("%8.2f");

        System.out.println("Macierz C");
        Matrix C = new Matrix(3, 2).random().printf("%8.2f");
        System.out.println("Macierz D");
        Matrix D = new Matrix(2, 4).random().printf("%8.2f");
        System.out.println("Iloczyn macierzy, C*D = ");
        C.mult(D).printf("%10.4f");

        System.out.println("Macierz C");
        C.printf("%10.4f");
        System.out.println("Iloczyn macierzy przez skalar, 3*C = ");
        C.mult(3).printf("%10.4f");
    }
}
```

Zadanie 28.29. Z28_29.java

Dołączamy do klasy `Matrix` metody zwracające macierz (kwadratową) reprezentowaną przez obiekt w postaci macierzy trójkątnej lub diagonalnej (na podstawie rozwiązania zadań 28.20, 28.21 i 28.22). Uzyskane macierze charakteryzują się tym, że mają taki sam wyznacznik jak macierz wyjściowa i jest on równy iloczynowi elementów z głównej przekątnej macierzy.

Macierz trójkątna górna — wszystkie elementy poniżej głównej przekątnej mają wartość zero.

```
public Matrix upperTriangular() {
    if (M.length != M[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = M.length;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < n-1; ++i)
        for(int j = i+1; j < n; ++j) {
            if (tmp.M[i][i] != 0.0) {
                double p = tmp.M[j][i]/tmp.M[i][i];
                for(int k = 0; k < n; ++k)
                    tmp.M[j][k] -= tmp.M[i][k]*p;
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp.M[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp.M[i][m];
                            tmp.M[i][m] = tmp.M[k][m];
                            tmp.M[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
    return tmp;
}
```

Macierz trójkątna dolna — wszystkie elementy poniżej głównej przekątnej mają wartość zero.

```
public Matrix lowerTriangular() {
    if (M.length != M[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = M.length;
    Matrix tmp = new Matrix(this);
    for(int i = n-1; i > 0; --i)
        for(int j = i-1; j >= 0; --j) {
            if (tmp.M[i][i] != 0.0) {
                double p = tmp.M[j][i]/tmp.M[i][i];
                for(int k = 0; k < n; ++k)
                    tmp.M[j][k] -= tmp.M[i][k]*p;
            } else {
                for(int k = i-1; k >= 0; --k) {
                    if (tmp.M[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp.M[i][m];
                            tmp.M[i][m] = tmp.M[k][m];
                            tmp.M[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
    return tmp;
}
```

Macierz diagonalna — wszystkie elementy poza główną przekątną mają wartość zero.

```
public Matrix diagonal() {
    if (M.length != M[0].length)
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            throw new ArithmeticException("To nie jest macierz kwadratowa");
        int n = M.length;
        Matrix tmp = new Matrix(this);
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j) {
                if (tmp.M[i][i] != 0.0) {
                    if (j != i) {
                        double p = tmp.M[j][i]/tmp.M[i][i];
                        for(int k = 0; k < n; ++k)
                            tmp.M[j][k] -= tmp.M[i][k]*p;
                    }
                } else {
                    for(int k = i+1; k < n; ++k) {
                        if (tmp.M[k][i] != 0.0) {
                            for(int m = 0; m < n; ++m) {
                                double p = tmp.M[i][m];
                                tmp.M[i][m] = tmp.M[k][m];
                                tmp.M[k][m] = -p;
                            }
                            break;
                        }
                    }
                }
            }
        return tmp;
    }
```

Macierz kwadratową o wylosowanych elementach przekształcamy na postać trójkątną (górną lub dolną) i postać diagonalną.

```
class Z28_29 {
    public static void main(String[] args) {
        System.out.println("Macierz A");
        Matrix A = new Matrix(5, 5).random().printf("%8.2f");

        System.out.println("Macierz trójkątna górna");
        A.upperTriangular().printf("%8.2f");

        System.out.println("Macierz trójkątna dolna");
        A.lowerTriangular().printf("%8.2f");

        System.out.println("Macierz diagonalna");
        A.diagonal().printf("%8.2f");
    }
}
```

Możemy zbudować dla macierzy trójkątnych lub diagonalnych metodę obliczającą wyznacznik macierzy:

```
private static double determinant(Matrix x) {
    /* Tylko dla macierzy kwadratowych - trójkątnych i diagonalnych */
    int n = x.getRows();
    double tmp = 1.0;
    for(int i = 0; i < n; ++i)
        tmp *= x.getAt(i, i);
    return tmp;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Demonstrując działanie metody, możemy obliczyć wyznacznik macierzy:

```
System.out.println("Macierz trójkątna górna");
System.out.printf("det A = %.4f\n",
    determinant(A.upperTriangular().printf("%8.2f")));
```

Zwróćmy uwagę na kolejność działań:

- `A.upperTriangular()` — zamieniamy macierz `A` na macierz trójkątną,
- `A.upperTriangular().printf("%8.2f")` — zamieniamy macierz `A` na macierz trójkątną i wyświetlamy ją w konsoli,
- `determinant(A.upperTriangular().printf("%8.2f")))` — zamieniamy macierz `A` na macierz trójkątną, wyświetlamy ją w konsoli i obliczamy wyznacznik (następnie możemy go wyświetlić lub wykorzystać w innych obliczeniach).

Oczywiście możemy skoncentrować się wyłącznie na obliczeniu wyznacznika, np.:

```
double w = determinant(A.lowerTriangular());
```

Wykonane operacje naturalnie nie zmieniają macierzy (obiektu) `A`.

Zadanie 28.30. Z28_30.java

Możemy skorzystać z rozwiązania zadania 28.19. Przedstawiona tam metoda rekurencyjna pochłania dużo pamięci, co ogranicza również stopień macierzy obliczanego wyznacznika. Ponadto zapożyczymy pomysł z zadania 28.29 i odpowiednio zmodyfikujemy metodę `upperTriangle()`, dostosowując ją do obliczania wyznacznika.

```
public double det() {
    if (M.length != M[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = M.length;
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < n-1; ++i) {
        for(int j = i+1; j < n; ++j) {
            if (tmp.M[i][i] != 0.0) {
                double p = tmp.M[j][i]/tmp.M[i][i];
                for(int k = 0; k < n; ++k)
                    tmp.M[j][k] -= tmp.M[i][k]*p;
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp.M[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp.M[i][m];
                            tmp.M[i][m] = tmp.M[k][m];
                            tmp.M[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        double d = 1.0;
        for(int i = 0; i < n; ++i)
            d *= tmp.M[i][i];
        return d;
    }
```

Działanie metody sprawdzimy na przykładzie macierzy z wylosowanymi wartościami elementów:

```
class Z28_30 {
    public static void main(String[] args) {
        System.out.println("Macierz A");
        Matrix A = new Matrix(5, 5).random().mult(10).printf("%8.2f");
        System.out.printf("Wyznacznik det A = %f\n", A.det());
    }
}
```

Zadanie 28.31. Z28_31.java

Na podstawie rozwiązania zadania 28.23 budujemy w klasie `Matrix` metodę `inverse()` zwracającą macierz odwrotną do macierzy (kwadratowej) reprezentowanej przez obiekt.

```
public Matrix inverse() {
    if (M.length != M[0].length)
        throw new ArithmeticException("To nie jest macierz kwadratowa");
    int n = M.length;
    Matrix tmp = new Matrix(this);
    Matrix I = new Matrix(n, n);
    for(int i = 0; i < n; ++i)
        I.M[i][i] = 1.0;

    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j) {
            if (tmp.M[i][i] != 0.0) {
                if (j != i) {
                    double p = tmp.M[j][i]/tmp.M[i][i];
                    for(int k = 0; k < n; ++k) {
                        tmp.M[j][k] -= tmp.M[i][k]*p;
                        I.M[j][k] -= I.M[i][k]*p;
                    }
                }
            } else {
                for(int k = i+1; k < n; ++k) {
                    if (tmp.M[k][i] != 0.0) {
                        for(int m = 0; m < n; ++m) {
                            double temp = tmp.M[i][m];
                            tmp.M[i][m] = tmp.M[k][m];
                            tmp.M[k][m] = -temp;
                            temp = I.M[i][m];
                            I.M[i][m] = I.M[k][m];
                            I.M[k][m] = -temp;
                        }
                        break;
                    }
                }
            }
        }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        for(int i = 0; i < n; ++i)
            if (tmp.M[i][i] != 0.0)
                for(int j = 0; j < n; ++j)
                    I.M[i][j] /= tmp.M[i][i];
            else
                throw new ArithmeticException("Macierz odwrotna nie
                    istnieje");
        return I;
    }
```

W przykładzie macierz kwadratową `A` wypełniamy wartościami losowymi i wyznaczamy macierz odwrotną `B` ($B = A^{-1}$). Poprawność obliczeń sprawdzamy, obliczając iloczyny macierzy $A \cdot B$ i $B \cdot A$. W obu przypadkach powinniśmy otrzymać macierz jednostkową. Wyznaczniki macierzy odwrotnych powinny być liczbami odwrotnymi i ich iloczyn powinien wynosić *1*. Uzyskanie takich wyników potwierdzi poprawność zbudowanych metod.

```
class Z28_31 {
    public static void main(String[] args) {
        System.out.println("Macierz A");
        Matrix A = new Matrix(5, 5).random().mult(10).printf("%8.2f");
        System.out.println("Macierz odwrotna B");
        Matrix B = A.inverse().printf("%8.2f");
        System.out.println("Iloczyn macierzy A*B =");
        A.mult(B).printf("%8.2f");
        System.out.println("Iloczyn macierzy B*A =");
        B.mult(A).printf("%8.2f");
        System.out.printf("Wyznacznik det A = %f\n", A.det());
        System.out.printf("Wyznacznik det B = %f\n", B.det());
        System.out.printf("det A * det B = %f\n", A.det()*B.det());
    }
}
```

Zadanie 28.32. Z28_32.java

Układ *n* równań liniowych o *n* niewiadomych:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

możemy zapisać w postaci równania macierzowego ($A \cdot X = B$):

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Mnożąc równanie lewostronnie przez macierz odwrotną do macierzy podstawowej układu, uzyskamy rozwiązanie układu ($X = A^{-1} \cdot B$):

Ebookpoint.pl kopia dla: pawher@o2.pl

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}^{-1} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Wyrażenie to możemy w łatwy sposób zapisać przy użyciu obiektów i metod klasy `Matrix` (`X = A.inverse().mult(B)`). Podobnie wykonamy sprawdzenie, obliczając `A.mult(X)` i porównując z kolumną wyrazów wolnych `B` (wizualnie, po wyświetleniu wyników w konsoli lub po zbudowaniu metody `equals()` w klasie `Matrix`).

```
class Z28_32 {
    public static void main(String[] args)  {
        /* Układ równań - macierz odwrotna */
        Matrix A = new Matrix(3, 3).random();
        Matrix B = new Matrix(3, 1).random();
        Matrix X;
        System.out.println("Układ równań A*X = B");
        System.out.println("Macierz główna A = ");
        A.printf("%8.2f");
        System.out.println("Kolumna wyrazów wolnych B = ");
        B.printf("%8.2f");
        System.out.println("Rozwiązanie układu X = ");
        X = A.inverse().mult(B).printf("%8.2f");
        System.out.println("Sprawdzenie A*X = ");
        A.mult(X).printf("%8.2f");
    }
}
```

Zadanie 28.33. Z28_33.java

Do klasy `Matrix` dołączmy metodę `replaceCol()`. Metoda ta wykonuje kopię obiektu (macierzy) i podstawia w miejscu wskazanej kolumny wartości elementów z jednokolumnowej macierzy przekazanej jako parametr (liczba wierszy obu macierzy musi być równa).

```
public Matrix replaceCol(int k, Matrix c) {
    if (k < 0 || k > M[0].length-1 || c.M.length != M.length
        || c.M[0].length != 1)
        throw new ArithmeticException("Niewłaściwe dane");
    int n = M.length;
    Matrix tmp = new Matrix(this);
    for(int i = 0; i < n; ++i)
        tmp.M[i][k] = c.M[i][0];
    return tmp;
}
```

Zdefiniowaną metodę `replaceCol()` wykorzystamy do rozwiązania układu równań metodą wyznaczników. Obliczmy wyznacznik główny `double w = A.det()`. Jeśli wyznacznik główny jest różny od zera, to układ jest oznaczony. W pętli obliczamy wyznaczniki dla kolejnych zmiennych — zastępujemy kolumnę współczynników przy tej zmiennej przez kolumnę wyrazów wolnych, liczymy wyznacznik i wartość niewiadomej (`Matrix Wx = new Matrix(A).replaceCol(i, B); X.setAt(i, 0, Wx.det()); X.setAt(i, 0, X.getAt(i, 0)/w);`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
class Z28_33 {
    public static void main(String[] args)  {
        /* Układ równań - metoda wyznaczników */
        int n = 4;
        Matrix A = new Matrix(n, n).random().mult(10);
        Matrix B = new Matrix(n, 1).random().mult(5);
        Matrix X = new Matrix(n, 1);

        System.out.println("Macierz główna A = ");
        A.printf("%8.2f");
        double w = A.det();
        System.out.printf("Wyznacznik główny W = %.4f\n", w);

        System.out.println("Kolumna wyrazów wolnych B = ");
        B.printf("%8.2f");
        if (w != 0) {
            for(int i = 0; i < n; ++i) {
                System.out.printf("Macierz Wx[%d]\n", i);
                Matrix Wx = new Matrix(A).replaceCol(i, B).
                    printf("%8.2f");
                X.setAt(i, 0, Wx.det());
                System.out.printf("Wyznacznik Wx[%d] = %.4f\n",
                    i, X.getAt(i, 0));
                X.setAt(i, 0, X.getAt(i, 0)/w);
            }
            System.out.println("Rozwiązanie układu X = ");
            X.printf("%8.2f");
            System.out.println("Sprawdzenie A*X = ");
            A.mult(X).printf("%8.2f");
        } else
            System.out.println("Układ sprzeczny lub nieoznaczony");
    }
}
```

Możemy tu dostrzec pewne komplikacje wynikające z wyświetlania wyników pośrednich; bez tego rachunek mógłby wyglądać tak:

```
X.setAt(i, 0, new Matrix(A).replaceCol(i, B).det()/w)
```

Jeśli wyznacznik główny jest zerem, to wyświetliliśmy informację, że układ jest sprzeczny lub nieoznaczony. Czytelnik może ten fragment rozbudować o analizę wartości wszystkich wyznaczników i orzeczenie, czy układ jest sprzeczny (co najmniej jeden wyznacznik różny od zera), czy nieoznaczony (wszystkie wyznaczniki równe zero).

29. Obliczanie wartości funkcji, rekurencja i inne zadania

Zadanie 29.1. Z29_1.java

Po uruchomieniu programu dla przykładowych danych zawartych w tablicy `x` zauważymy, że metoda `f()` zwraca bezwzględną wartość liczby. Metodę tę możemy nazwać

Ebookpoint.pl kopia dla: pawher@o2.pl

skrótem `abs` (jak postąpilibyśmy w większości języków programowania), pochodzącym od angielskiej nazwy ***abs**olute value* (bezwzględna wartość liczby).

```
class Z29_1 {
    public static double f(double x) {
        return (x > 0)?x:-x;
    }

    public static void main(String[] args) {
        double[] x = {-3.3, -2.15, -1.0, 0.0, 1.25, 3,1415, 7.5};
        for(double a: x)
            System.out.println("a = "+a+",\tf(a) = "+f(a));
    }
}
```

Jedynie dla liczby *0,0* zamiast spodziewanej wartości *0,0* otrzymamy *–0,0*. Wynika to z istnienia w typach liczb zmiennoprzecinkowych dwóch reprezentacji zera. Wspomnianą niedogodność (dotyczącą zera) możemy wyeliminować, modyfikując nieznacznie kod metody:

```
static double abs(double x) {
    return (x >= 0)?x:-x;
}
```

Zadanie 29.2. MinMax.java, Z29_2.java

Budujemy klasę `MinMax` zawierającą metody `min()` i `max()` z dwoma parametrami tego samego typu (`double`, `float`, `int` lub `long`):

```
class MinMax {
    public static double min(double x, double y) {
        if (x <= y)
            return x;
        else
            return y;
    }
    public static double max(double x, double y) {
        if (x >= y)
            return x;
        else
            return y;
    }
    /* pozostałe metody... */
}
```

Zamiast instrukcji warunkowej możemy zastosować wyrażenie warunkowe, co skróci zapis kodu źródłowego:

```
public static double min(double x, double y) {
    return (x <= y)?x:y;
}
public static double max(double x, double y) {
    return (x >= y)?x:y;
}
```

Podobnie możemy zmodyfikować kod pozostałych metod.

Ebookpoint.pl kopia dla: pawher@o2.pl

Działanie metod `min()` i `max()` możemy sprawdzić w programie:

```
class Z29_2 {
    public static void main(String[] args) {
        /* Liczby typu double */
        System.out.println(MinMax.min(2.5, -3.2));
        System.out.println(MinMax.min(2.5, 5.2));
        /* Liczby typu float */
        float a = MinMax.min(-5.66f, 2.0f);
        System.out.println(a);
        System.out.println(MinMax.max(a, a/2));
        /* ... */
    }
}
```

Zadanie 29.3. Z29_3.java

Funkcja *f(x)* zwraca znak liczby: *–1* dla liczb ujemnych, *0* dla *0* i *1* dla liczb dodatnich. Odpowiednią nazwą mogłoby być słowo `znak`, łac. `signum` lub stosowany w matematyce skrót `sgn`.

Kod metody `sgn()` obliczającej wartość tej funkcji możemy zapisać, używając instrukcji warunkowych:

```
static int sgn(double x) {
    if (x < 0)
        return -1;
    else if (x == 0.0)
        return 0;
    else
        return 1;
}
```

lub wyrażenia warunkowego:

```
static int sgn(double x) {
    return (x < 0)?-1:(x == 0)?0:1;
}
```

W taki sam sposób zdefiniujemy metodę `sgn()` dla innych typów argumentów.

```
public class Z29_3 {
    /* Tu wstaw kod metody sgn(). */
    public static void main(String[] args) {
        System.out.println("Znak liczby -3.56: "+sgn(-3.56));
        System.out.println("Znak liczby 0.0: "+sgn(0.0));
        System.out.println("Znak liczby 4.2: "+sgn(4.2f));
    }
}
```

Zadanie 29.4. Z29_4.java

Wyrażenie $\frac{|x|}{x}$ przyjmuje wartość 1 dla $x > 0$ i –1 dla $x < 0$. Zatem funkcja $f(x)$ zwraca znak liczby (zob. rozwiązanie zadania 29.3). Metodę obliczającą wartość tej funkcji możemy nazwać `sgn` (skrót od łac. *signum*):

Ebookpoint.pl kopia dla: pawher@o2.pl

```
static int sgn(double x) {
    if (x == 0.0)
        return 0;
    else
        return (int)(Math.abs(x)/x);
}
```

lub:

```
static int sgn(double x) {
    return ((x == 0.0)?0:(int)(Math.abs(x)/x));
}
```

W podobny sposób zdefiniujemy metody dla innych typów argumentów. Ponieważ wynik powinien być typu `int`, a wartość wyrażenia `Math.abs(x)/x` jest zgodna z typem argumentu `x`, musimy pamiętać o rzutowaniu tej wartości na typ `int`.

Zadanie 29.5. Z29_5.java

a) Funkcja $f(x, y)$ zwraca maksimum dwóch liczb `x` i `y` i możemy tę funkcję oraz odpowiadającą jej metodę oznaczyć skrótem `max`.

```
public static double f(double x, double y) {
    return (x+y+Math.abs(x-y))/2;
}
```

b) Funkcja $g(x, y)$ zwraca minimum dwóch liczb `x` i `y` i możemy tę funkcję oraz odpowiadającą jej metodę oznaczyć skrótem `min`.

```
public static double g(double x, double y) {
    return (x+y-Math.abs(x-y))/2;
}
```

c) Wśród liczb `x` i `-x` większą jest liczba dodatnia, a zatem metoda `h()` zwraca bezwzględną wartość liczby `x`.

```
public static double h(double x) {
    return f(x, -x);
}
```

Test działania metod `f()` i `g()` przeprowadzimy na *9* parach liczb, natomiast metodę `h()` przetestujemy dla *6* wartości.

```
class Z29_5 {
    /* Tu wstaw kod funkcji f, g, h. */
    public static void main(String[] args) {
        double[] a = {-3.3, 0.0, 1.25};
        double[] b = {-5.12, 0.25, 1.5};
        for(double x: a)
           for(double y: b) {
               System.out.printf("f(%f, %f) = %f\n", x, y, f(x, y));
               System.out.printf("g(%f, %f) = %f\n", x, y, g(x, y));
           }
        for(double x: a)
            System.out.printf("h(%f) = %f\n", x, h(x));
        for(double x: b)
            System.out.printf("h(%f) = %f\n", x, h(x));
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 29.6. Z29_6.java

Na podstawie wzorów $x^2 = x \cdot x$ i $x^3 = x \cdot x \cdot x$ zbudujemy metody `square()` (kwadrat) i `cube()` (sześcian). Metody tworzymy w dwóch wersjach: dla argumentów zmiennoprzecinkowych i dla argumentów całkowitych.

```
class Z29_6 {
    public static double square(double x) {
        return x*x;
    }
    public static double cube(double x) {
        return x*x*x;
    }
    public static int square(int x) {
        return x*x;
    }
    public static int cube(int x) {
        return x*x*x;
    }

    public static void main(String[] args) {
        System.out.println("Kwadraty i sześciany liczb od 1 do 15");
        for(int n = 1; n < 16; ++n)
            System.out.printf("%3d%8d%8d\n", n, square(n), cube(n));
        System.out.println();

        System.out.println("Kwadraty i sześciany liczb od 1.0 do 3.0");
        double x = 1.0;
        while (x <= 3.0) {
            System.out.printf("%7.2f%10.4f ", x, square(x));
            System.out.printf("%12.6f\n", cube(x));
            x += 0.25;
        }
    }
}
```

Zadanie 29.7. Z29_7.java

Ponieważ $x^0 = 1$, to przyjmujemy 1 jako początkową wartość potęgi (`double p = 1.0`). Następnie w pętli *n*-krotnie mnożymy tę wartość przez podstawę (`p *= x`).

```
class Z29_7 {
    public static double adoen(double x, int n) {
        double p = 1.0;
        for(int i = 0; i < n; ++i)
            p *= x;
        return p;
    }
    public static void main(String[] args) {
        System.out.println("Potęgi liczby 2");
        for(int n = 0; n <= 10; ++n)
            System.out.println(adoen(2, n));
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Wyniki zostaną wyświetlone w postaci ułamka dziesiętnego, np. *1.0*, mimo że są to liczby całkowite. Możemy tę niedogodność wyeliminować, stosując rzutowanie wyniku na typ całkowity `System.out.println((int)adoen(2, n));` lub definiując metodę `adoen()` dla podstawy typu całkowitego:

```
public static int adoen(int x, int n) {
    int p = 1;
    for(int i = 0; i < n; ++i)
        p *= x;
    return p;
}
```

Zadanie 29.8. Z29_8.java

Zdefiniujmy metody rekurencyjne `adoen()` dla podstawy zmiennoprzecinkowej:

```
public static double adoen(double x, int n) {
    if (n == 0)
        return 1.0;
    else
        return x*adoen(x, n-1);
}
```

i dla podstawy całkowitej:

```
public static int adoen(int x, int n) {
    if (n == 0)
        return 1;
    else
        return x*adoen(x, n-1);
}
```

W obu przypadkach wykładnik może być dowolną liczbą naturalną.

Działanie metody możemy sprawdzić, obliczając kilka potęg, np.:

```
System.out.println("Potęgi liczby 2");
for(int n = 0; n <= 10; ++n)
    System.out.println(power(2, n));
```

Zadanie 29.9. Z29_9.java

Zacznijmy od zdefiniowania metody `sqr()` obliczającej kwadrat liczby.

```
public static double sqr(double x) {
    return x*x;
}
```

Zastosowanie metody `sqr()` zredukuje o połowę liczbę wywołań rekurencyjnych metody `adoen()`.

```
public static double adoen(double x, int n) {
    if (n == 0)
        return 1;
    else if (n%2 == 1) // n jest nieparzyste
        return x*sqr(adoen(x, (n-1)/2));
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    else                    // n jest parzyste
        return sqr(adoen(x, n/2));
}
```

Działanie metody możemy sprawdzić, np. wykonując obliczenia kilku wartości potęg liczby 2.

```
for(int i = 0; i <= 10; ++i)
    System.out.println(adoen(2, i));
```

Aby obliczać potęgi o podstawie całkowitej, możemy zdefiniować analogiczne metody, stosując typ `int` lub `long`.

Zadanie 29.10. Z29_10.java

Metodę `power()` możemy zbudować, wykorzystując instrukcję warunkową:

```
public static double power(double x, int n) {
    if (n < 0)
        return 1/adoen(x, -n);
    else
        return adoen(x, n);
}
```

lub wyrażenie warunkowe:

```
public static double power(double x, int n) {
    return (n < 0)?1/adoen(x, -n):adoen(x, n);
}
```

Metodę `adoen()` utworzymy na podstawie rozwiązania zadania 29.7, 29.8 lub 29.9 (w tym ostatnim przypadku potrzebna będzie metoda `sqr()`). Działanie funkcji sprawdzimy, wykonując obliczenia kilku wartości potęg liczby 2.

```
for(int i = -10; i <= 10; ++i)
    System.out.println(power(2, i));
```

Zadanie 29.11. Z29_11.java

Na podstawie wzoru iteracyjnego $x = \frac{1}{2}\left(x + \frac{a}{x}\right)$ budujemy metodę `sqrt()`. W pętli przechowujemy w zmiennej `pw` poprzednią wartość przybliżenia `x` i obliczamy kolejną iterację. Obliczenia kontynuujemy, gdy różnica (bezwzględna wartość różnicy) jest dostatecznie duża (większa lub równa *0,00000001*). Precyzję obliczeń możemy zmienić w zależności od potrzeb.

```
class Z29_11 {
    public static double sqrt(double a) {
        double pw, x = 1;
        do {
            pw = x;
            x = (x+a/x)/2;
        } while (Math.abs(pw-x) >= 0.00000001);
        return x;
    }
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
    public static void main(String[] args) {
        System.out.println("Pierwiastki kwadratowe");
        for(int i = 1; i <= 20; ++i) {
            System.out.printf("%3d%15.8f", i, sqrt(i));
            System.out.printf("%15.8f\n", Math.sqrt(i));
        }
    }
}
```

Testując działanie metody `sqrt()`, obliczyliśmy pierwiastki kwadratowe z liczb od 1 do 20. Wyniki wyświetliliśmy z dokładnością do 8 miejsc po przecinku. Są one zgodne z wynikami zwróconymi przez metodę biblioteczną `Math.sqrt()`.

Zadanie 29.12. Z29_12.java

Rozwiązanie zadania jest całkowicie podobne do rozwiązania zadania 29.11. Należy jedynie poprawić nazwy funkcji i zmienić wzór iteracyjny.

```
class Z29_12 {
    public static double cbrt(double a) {
        double pw, x = 1;
        do {
            pw = x;
            x = (2*x+a/(x*x))/3;
        } while (Math.abs(pw-x) >= 0.00000001);
        return x;
    }

    public static void main(String[] args) {
        System.out.println("Pierwiastki sześcienne");
        for(int i = 1; i <= 20; ++i) {
            System.out.printf("%3d%15.8f", i, cbrt(i));
            System.out.printf("%15.8f\n", Math.cbrt(i));
        }
    }
}
```

Zadanie 29.13. Z29_13.java

Rozwiązanie zadania jest uogólnieniem przypadków przedstawionych w zadaniach 29.11 i 29.12. Ze względu na potrzebę częstego obliczania potęg o wykładniku całkowitym i podstawie dodatniej wykorzystamy tu jedną z wersji metody `adoen()` (zob. zadanie 29.7, 29.8 lub 29.9). Uzyskane wyniki porównamy z wynikami zwróconymi przez standardową metodę `Math.pow()`.

```
class Z29_13 {
    public static double adoen(double x, int n) {
    double p = 1.0;
    for(int i = 0; i < n; ++i)
        p *= x;
    return p;
    }

    public static double nRoot(double a, int n) {
        double pw, x = 1;
        do {
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
            pw = x;
            x = ((n-1)*x+a/adoen(x, n-1))/n;
        } while (Math.abs(pw-x) >= 0.00000001);
        return x;
    }

    public static void main(String[] args) {
        System.out.println("Pierwiastki n-tego stopnia z 8");
        for(int n = 2; n <= 10; ++n) {
            System.out.printf("%3d%15.8f", n, nRoot(8, n));
            System.out.printf("%15.8f\n", Math.pow(8, 1.0/n));
        }
    }
}
```

Zadanie 29.14. Z29_14.java

Oprócz *sinusa* i *cosinusa hiperbolicznego* (metody obliczające wartość tych funkcji są dostępne w klasie `Math`) mamy cztery inne funkcje hiperboliczne:

- *tangens hiperboliczny* $\tanh x = \frac{\sinh x}{\cosh x}$ (inne oznaczenia: tgh x lub th x),
- *cotangens hiperboliczny* $\coth x = \frac{\cosh x}{\sinh x} = \frac{1}{\tanh x}$ (inne oznaczenia: ctgh x lub cth x),
- *secans hiperboliczny* $\operatorname{sech} x = \frac{1}{\cosh x}$,
- *cosecans hiperboliczny* $\operatorname{csch} x = \frac{1}{\sinh x}$.

```
class Z29_14{
    static String[] title = {"  x  ", "sinh x", "cosh x", "tanh x",
                             "coth x", "sech x", "csch x"};
    public static void main(String[] args) {
        System.out.println("Tablice funkcji hiperbolicznych\n");
        for(String str: title)
            System.out.printf("%9s", str);
        System.out.println();
        double x, sh, ch, th;
        for(int n = -50; n <= 50; ++n) {
            x = (double)n/10;
            sh = Math.sinh(x);   // sinus hiperboliczny
            ch = Math.cosh(x);   // cosinus hiperboliczny
            th = sh/ch;          // tangens hiperboliczny
            System.out.printf("%9.1f%9.4f%9.4f%9.4f", x, sh, ch, th);
            // cotangens, secans i cosecans hiperboliczny
            System.out.printf("%9.4f%9.4f%9.4f\n", 1/th, 1/ch, 1/sh);
        }
    }
}
```

Skompilowany program uruchamiamy poleceniem

```
java Z29_14 > FunkcjeHiperboliczne.txt
```

i wyniki zostaną przekierowane z konsoli do pliku tekstowego.

Ebookpoint.pl kopia dla: pawher@o2.pl

Zadanie 29.15. Z29_15.java

Wprowadzimy dodatkowe zmienne: `ex1 = Math.exp(x)` (e^x) i `ex2 = 1/ex1` ($\frac{1}{e^x} = e^{-x}$).

Na podstawie wzorów $\sinh x = \frac{e^x - e^{-x}}{2}$ i $\cosh x = \frac{e^x + e^{-x}}{2}$ obliczymy `sh = (ex1-ex2)/2` i `ch = (ex1+ex2)/2`. Dalsze obliczenia przeprowadzimy tak samo jak w zadaniu 29.14.

```
double x, ex1, ex2, sh, ch, th;
for(int n = -50; n <= 50; ++n) {
    x = (double)n/10;
    ex1 = Math.exp(x);
    ex2 = 1/ex1;
    sh = (ex1-ex2)/2;   // sinus hiperboliczny
    ch = (ex1+ex2)/2;   // cosinus hiperboliczny
    th = sh/ch;         // tangens hiperboliczny
    System.out.printf("%9.1f%9.4f%9.4f%9.4f", x, sh, ch, th);
    // cotangens, secans i cosecans hiperboliczny
    System.out.printf("%9.4f%9.4f%9.4f\n", 1/th, 1/ch, 1/sh);
}
```

Jeśli wprowadzimy dodatkowe zmienne `r=ex1-ex2` i `s=ex1+ex2`, to dalej możemy obliczyć `sh=r/2; ch=s/2; th=r/s;`. Wyrażenia obliczające wartości kolejnych funkcji (*cotangens*, *secans* i *cosecans hiperboliczny*) przyjmą postać: `s/r`, `2/s`, `2/r`:

```
double x, ex1, ex2, r, s, sh, ch, th;
for(int n = -50; n <= 50; ++n) {
    x = (double)n/10;
    ex1 = Math.exp(x);
    ex2 = 1/ex1;
    r = ex1-ex2;
    s = ex1+ex2;
    sh = r/2;   // sinus hiperboliczny
    ch = s/2;   // cosinus hiperboliczny
    th = r/s;   // tangens hiperboliczny
    System.out.printf("%9.1f%9.4f%9.4f%9.4f", x, sh, ch, th);
    // cotangens, secans i cosecans hiperboliczny
    System.out.printf("%9.4f%9.4f%9.4f\n", s/r, 2/s, 2/r);
}
```

Wyniki obliczeń zapiszemy w pliku tekstowym (zob. rozwiązanie zadania 29.15).

Zadanie 29.16. FH.java, Z29_16.java

W klasie `FH` (`public class FH {}`) zdefiniujemy na podstawie rozwiązania zadania 29.14 lub 29.15 metody obliczające wartości *funkcji hiperbolicznych*:

```
static double sinh(double x) {/* sinus hiperboliczny */
    double ex = Math.exp(x);
    return (ex-1/ex)/2;
}

static double cosh(double x) {/* cosinus hiperboliczny */
    double ex = Math.exp(x);
    return (ex+1/ex)/2;
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
}

static double tanh(double x) {/* tangens hiperboliczny */
    double ex1 = Math.exp(x);
    double ex2 = 1/ex1;
    return (ex1-ex2)/(ex1+ex2);
}

static double coth(double x) {/* cotangens hiperboliczny */
    double ex1 = Math.exp(x);
    double ex2 = 1/ex1;
    return (ex1+ex2)/(ex1-ex2);
}

static double sech(double x) {/* secans hiperboliczny */
    double ex = Math.exp(x);
    return 2/(ex+1/ex);
}

static double csch(double x) {/* cosecans hiperboliczny */
    double ex = Math.exp(x);
    return 2/(ex-1/ex);
}
```

Funkcje odwrotne do funkcji hiperbolicznych możemy obliczyć na podstawie wzorów:

- *area sinus hiperboliczny* $\operatorname{arsinh} x = \ln\left(x + \sqrt{x^2+1}\right)$ — funkcja odwrotna do sinusa hiperbolicznego:

```
static double arsinh(double x) {/* area sinus hiperboliczny */
    return Math.log(x+Math.sqrt(x*x+1));
}
```

- *area cosinus hiperboliczny* $\operatorname{arcosh} x = \ln\left(x + \sqrt{x^2-1}\right)$ dla $x \geq 1$ — funkcja odwrotna do cosinusa hiperbolicznego:

```
static double arcosh(double x) {/* area cosinus hiperboliczny */
    return Math.log(x+Math.sqrt(x*x-1));
}
```

- *area tangens hiperboliczny* $\operatorname{artanh} x = \frac{1}{2}\ln\frac{1+x}{1-x}$ — funkcja odwrotna do tangensa hiperbolicznego:

```
static double artanh(double x) {/* area tangens hiperboliczny */
    return Math.log((1+x)/(1-x))/2;
}
```

- *area cotangens hiperboliczny* $\operatorname{arcoth} x = \frac{1}{2}\ln\frac{x+1}{x-1}$ — funkcja odwrotna do cotangensa hiperbolicznego:

```
static double arcoth(double x) {/* area cotangens hiperboliczny */
    return Math.log((x+1)/(x-1))/2;
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

♦ *area secans hiperboliczny* $\operatorname{arsech} x = \ln\left(\sqrt{\frac{1}{x^2}-1}+\frac{1}{x}\right)$ — funkcja odwrotna do secansa hiperbolicznego:

```
static double arsech(double x) {/* area secans hiperboliczny */
    return Math.log(Math.sqrt(1/(x*x)-1)+1/x);
}
```

♦ *area cosecans hiperboliczny* $\operatorname{arcsch} x = \ln\left(\sqrt{1+\frac{1}{x^2}}+\frac{1}{x}\right)$ — funkcja odwrotna do cosecansa hiperbolicznego:

```
static double arcsch(double x) {/* area cosecans hiperboliczny */
    return Math.log(Math.sqrt(1+1/(x*x))+1/x);
}
```

Demonstrując działanie zdefiniowanych metod, obliczymy wartości funkcji hiperbolicznych (w pętli) dla kilku wybranych argumentów, jednocześnie dla uzyskanych wyników obliczymy wartości funkcji odwrotnej (obliczenia zestawimy w trzech kolumnach). Porównując pierwszą i trzecią kolumnę, sprawdzimy poprawność obliczeń.

```
System.out.println("Sinus hiperboliczny i area sinus hiperboliczny");
double x = -7.05, y;
while (x < 10) {
    y = FH.sinh(x);
    System.out.printf("%15.7f%15.7f%15.7f\n", x, y, FH.arsinh(y));
    x += 1.47;
}
```

Podobne obliczenia przeprowadzimy dla pozostałych funkcji hiperbolicznych i funkcji do nich odwrotnych.

Zadanie 29.17. FTD.java, 29_17.java

W klasie `FTD` (`public class FTD {}`) zdefiniujemy metody obliczające wartości funkcji trygonometrycznych, w których argumentem jest miara kąta wyrażona w stopniach. Wykorzystamy w tym celu metody zdefiniowane w klasie `Math` (`sin()`, `cos()` i `tan()`) oraz stałą `Math.PI`.

```
static double sin(double x) { /* sinus */
    return Math.sin(x/180*Math.PI);
}

static double cos(double x) { /* cosinus */
    return Math.cos(x/180*Math.PI);
}

static double tan(double x) { /* tangens */
    return Math.tan(x/180*Math.PI);
}
```

Wyrażenie `x/180*Math.PI` przeliczające stopnie na radiany możemy zastąpić wywołaniem metody `Math.toRadians(x)`.

Ebookpoint.pl kopia dla: pawher@o2.pl

Na podstawie zależności $\cot x = \frac{1}{\tan x}$ (*cotangens*), $\sec x = \frac{1}{\cos x}$ (*secans*) i $\csc x = \frac{1}{\sin x}$ (*cosecans*) zdefiniujemy:

```
static double cot(double x) { /* cotangens */
    return 1/Math.tan(x/180*Math.PI);
}

static double sec(double x) { /* secans */
    return 1/Math.cos(x/180*Math.PI);
}

static double csc(double x) { /* cosecans */
    return 1/Math.sin(x/180*Math.PI);
}
```

W klasie `Math` zdefiniowano trzy metody obliczające wartości funkcji odwrotnych do funkcji trygonometrycznych: *arcus sinus* (`asin()`), *arcus cosinus* (`acos()`) i *arcus tangens* (`atan()`), zwracające wynik (miarę kąta) wyrażony w radianach. Dzieląc wynik przez π i mnożąc przez 180, otrzymamy miarę kąta w stopniach, np. `Math.asin(x)/Math.PI*180`. Podane wyrażenie możemy zastąpić wywołaniem metody `toDegrees()` z klasy `Math` (`Math.toDegrees(Math.asin(x))`).

```
static double arcsin(double x) {
    return Math.asin(x)/Math.PI*180;
}

static double arccos(double x) {
    return Math.acos(x)/Math.PI*180;
}

static double arctan(double x) {
    return Math.atan(x)/Math.PI*180;
}
```

Wykorzystamy zależności $\tan x = \frac{1}{\cot x}$, $\cos x = \frac{1}{\sec x}$ i $\sin x = \frac{1}{\csc x}$ do wyznaczenia funkcji odwrotnych dla funkcji *cotangens*, *secans* i *cosecans*.

```
static double arccot(double x) {
    return Math.atan(1/x)/Math.PI*180;
}

static double arcsec(double x) {
    return Math.acos(1/x)/Math.PI*180;
}

static double arccsc(double x) {
    return Math.asin(1/x)/Math.PI*180;
}
```

Do klasy dołączymy dwie metody zamieniające miary kątów: stopnie na radiany (`degToRad()`) i radiany na stopnie (`radToDeg()`).

Ebookpoint.pl kopia dla: pawher@o2.pl

```
static double degToRad(double x) {
    return x/180*Math.PI;
}

static double radToDeg(double x) {
    return x/Math.PI*180;
}
```

Demonstrując działanie metod z klasy FTD, zbudujemy tablicę funkcji trygonometrycznych dla wybranych kątów (o mierze wyrażonej w stopniach).

```
System.out.println("Funkcje trygonometryczne\n");
String[] title = {"x", "sin x", "cos x", "tan x", "cot x", "sec x", "csc x"};
for(String s:title)
    System.out.printf("%10s", s);
System.out.println();
double[] alfa = {0.1, 1.0, 15.0, 30.0, 45.0, 60.0, 75.0, 89.0, 89.9};
for(double x: alfa) {
    System.out.printf("%10.1f%10.4f%10.4f", x, FTD.sin(x), FTD.cos(x));
    System.out.printf("%10.4f%10.4f", FTD.tan(x), FTD.cot(x));
    System.out.printf("%10.4f%10.4f\n", FTD.sec(x), FTD.csc(x));
}
```

Testując działanie funkcji odwrotnych, obliczymy miary kątów ostrych w tzw. *trójkącie egipskim* (trójkąt prostokątny o bokach długości 3, 4 i 5 jednostek).

```
System.out.println("\nFunkcje odwrotne do funkcji trygonometrycznych");
System.out.println("Obliczanie miary kąta ostrego w trójkącie egipskim");
System.out.println("(a = 3, b = 4 i c = 5)\n");
double a = 3.0, b = 4.0, c = 5.0; // boki trójkąta egipskiego
System.out.println("sin A = "+a/c+",\tA = "+FTD.arcsin(a/c)+"°");
System.out.println("cos A = "+b/c+",\tA = "+FTD.arccos(b/c)+"°");
System.out.println("tan A = "+a/b+",\tA = "+FTD.arctan(a/b)+"°");
System.out.println("cot A = "+b/a+",\tA = "+FTD.arccot(b/a)+"°");
System.out.println("sec A = "+c/b+",\tA = "+FTD.arcsec(c/b)+"°");
System.out.println("csc A = "+c/a+",\tA = "+FTD.arccsc(c/a)+"°");
```

Na koniec sprawdzimy, ile *stopni* ma *radian* i jaką częścią *radiana* jest *stopień*.

```
System.out.println("\nKonwersja miar kątów");
System.out.println("1 rad = "+FTD.degToRad(1)+"°");
System.out.println("1° = "+FTD.radToDeg(1)+" rad");
```

Zadanie 29.18. FTR.java, Z29_18.java

Możemy skopiować kod klasy FTD do pliku *FTR.java* i zmienić nazwę klasy na FTR. Nie zmieniamy kodu metod dokonujących zamiany miar kątów, natomiast w pozostałych metodach usuwamy wyrażenia zamieniające stopnie na radiany (z wyrażenia x/180*Math.PI pozostawiamy x) i radiany na stopnie (w metodach obliczających wartości funkcji odwrotnych usuwamy /Math.PI*180).

```
public class FTR {
    /* Funkcje trygonometryczne - miara kąta wyrażona w radianach */

    static double sin(double x) { /* sinus */
```

Ebookpoint.pl kopia dla: pawher@o2.pl

```
        return Math.sin(x);
    }
    /* itd. */

    /* Funkcje odwrotne - wynik w radianach */
    static double arcsin(double x) {
        return Math.asin(x);
    }
    /* itd. */

    /* Konwersje miar kątów */
    /* Kod pozostawiamy bez zmian. */
}
```

Przykładowe obliczenia wykonamy dla kąta o mierze 60°, zaczynając od zamiany miary stopniowej kąta na miarę łukową (w radianach):

```
public class Z29_18 {
    public static void main(String args[]) {
        double a = FTR.degToRad(60);
        System.out.println("a = "+a+" rad");
        System.out.println("sin a = "+FTR.sin(a));
        System.out.println("cos a = "+FTR.cos(a));
        System.out.println("tan a = "+FTR.tan(a));
        System.out.println("cot a = "+FTR.cot(a));
        System.out.println("sec a = "+FTR.sec(a));
        System.out.println("csc a = "+FTR.csc(a));
        System.out.println("a = "+FTR.radToDeg(a)+"\u00B0");
    }
}
```

Ebookpoint.pl kopia dla: pawher@o2.pl

Notatki

Ebookpoint.pl kopia dla: pawher@o2.pl

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

Ebookpoint.pl kopia dla: pawher@o2.pl