JAVA
OD PODSTAW

Tom 1 – od pierwszego programu do pisania metod

Instalacja Java
Pierwszy program
Komentarze i formatowanie kodu
Zmienne, stałe, typy prymitywne
Typ String
Operatory i rzutowanie
Instrukcje warunkowe
Pętle
Tablice
Metody

Przemysław Kruglej

Wydanie własne, pierwsze (04-2024)

Numer ISBN „Java od podstaw – Tom 1”: 978-83-968045-3-2
Numer ISBN kolekcji „Java od podstaw”: 978-83-968045-2-5

Kontakt z autorem: przemyslaw.kruglej@gmail.com

Strona Internetowa tej książki: https://kursjava.com/ksiazki/java-od-podstaw#tom-1

Kody źródłowe oraz rozwiązania zadań dostępne są na poniższej stronie w Internecie:

https://github.com/przemyslaw-kruglej/java-od-podstaw-przyklady

Errata dostępna jest w sekcji „Tom 1 – Wydanie pierwsze – 04-2024” na stronie:

https://github.com/przemyslaw-kruglej/java-od-podstaw-errata

7.8. Podsumowanie

7.8.1. Instrukcje warunkowe

✓ Instrukcje warunkowe służą do warunkowego wykonywania fragmentów kodu:

```
if (mianownik != 0) {
  System.out.println("Wynik: " + licznik / mianownik);
} else {
  System.out.println("Nie można dzielić przez zero!");
}
```

✓ Instrukcje warunkowe sprawdzają *prawdziwość* pewnego warunku bądź warunków, tzn. sprawdzają, czy warunek ma wartość `true` (prawda) czy `false` (fałsz). W przykładzie z licznikiem i mianownikiem:

- jeżeli `mianownik` jest różny od `0`, to wykonana zostanie instrukcja, która wypisuje na ekran wynik dzielenia zmiennej `licznik` przez `mianownik`,
- w przeciwnym razie wykonana zostanie instrukcja, która wypisze na ekran komunikat o błędnej wartości zmiennej `mianownik`.

✓ Instrukcje warunkowe mogą mieć wiele sekcji – kolejne zapisujemy za pomocą `else if`. Ponadto, możemy dodać jedną opcjonalną sekcję `else`, która wykonywana jest, gdy żaden z warunków nie będzie prawdziwy. Sekcja ta musi być zawsze na końcu instrukcji warunkowej:

```
if (x > 0) {
  System.out.println("x jest większe od 0.");
} else if (x < 0) {
  System.out.println("x jest mniejsze od 0.");
} else {
  System.out.println("x jest równe 0.");
}
```

✓ Warunki sprawdzane są zawsze w kolejności od pierwszego do ostatniego. Jeżeli pierwszy warunek nie zostanie spełniony, to sprawdzony będzie kolejny (o ile istnieje kolejna sekcja `else if`), aż znaleziony zostanie warunek, który będzie miał wartość `true`, lub żaden z warunków nie będzie miał wartości `true`. W takim przypadku wykonane zostaną instrukcje pod sekcją `else`, o ile jest obecna.

✓ Instrukcje warunkowe mogą obejmować wiele instrukcji – należy je wtedy ująć w nawiasy klamrowe `{}`.

✓ Jeżeli do sekcji instrukcji warunkowej przypisana jest tylko jedna instrukcja, to nie musimy używać nawiasów klamrowych. Jednakże, często stosowanym standardem jest ujmowanie nawet pojedynczych instrukcji w nawiasy klamrowe. Oba następujące przykłady są poprawne:

```
if (x > 0)
  System.out.println("x jest większe od 0.");
else if (x < 0)
  System.out.println("x jest mniejsze od 0.");
else
  System.out.println("x jest równe 0.");
```

```
if (x > 0) System.out.println("x jest większe od 0.");
else if (x < 0) System.out.println("x jest mniejsze od 0.");
else System.out.println("x jest równe 0.");
```

✓ Instrukcje warunkowe mogą być zagnieżdżone – wtedy klauzule `else` oraz `else if` odnoszą się do poprzedniego bloku instrukcji warunkowych:

```
if (operacja.equals("+")) { // 1
  wynik = liczba1 + liczba2;
} else if (operacja.equals("-")) {
  wynik = liczba1 - liczba2;
} else if (operacja.equals("*")) {
  wynik = liczba1 * liczba2;
} else if (operacja.equals("/")) {
  if (liczba2 != 0) { // 2
    wynik = liczba1 / liczba2;
  } else { // odnosi się do instrukcji (2)
    System.out.println("Nie można dzielić przez zero!");
  }
} else { // odnosi się do instrukcji (1)
  System.out.println("Nieprawidłowa operacja!");
}
```

✓ Instrukcje przypisane do danej sekcji warunkowej powinny mieć wcięcia (być wysunięte o np. dwie spacje), jak w przykładzie dotyczącym zagnieżdżonych instrukcji `if`. Instrukcje w zewnętrznej instrukcji `if` mają pojedyncze wcięcie. W zagnieżdżonej instrukcji `if` przyporządkowane do niej instrukcje wcięte są o kolejny poziom.

7.8.2. Operatory związane z instrukcjami warunkowymi

✓ Operatory relacyjne to operatory dwuargumentowe, które porównują wartości swoich argumentów, i zwracają jedną z wartości: `true` bądź `false`.

✓ Do dyspozycji mamy następujące operatory relacyjne:

- `<` mniejsze niż
- `>` większe niż

- `<=` mniejsze bądź równe
- `>=` większe bądź równe
- `==` równe (nie mylić z operatorem przypisania =)
- `!=` nierówne

✓ Operatory relacyjne są często używane do walidacji danych, czyli sprawdzenia, czy mają one poprawne wartości:

```
if (dzienTygodnia < 1) {
  System.out.println("Dzień tygodnia musi być >= 1.");
} else if (dzienTygodnia > 7) {
  System.out.println("Dzień tygodnia musi być <= 7.");
}
```

✓ Aby zapisać bardziej skomplikowany warunek, łączymy kolejne warunki przy użyciu operatora warunkowego `&&` (AND – i) bądź `||` (OR – lub). Warunki mogą być dowolnie złożone i używać każdego z operatorów dowolną liczbę razy. Przykład złożonego warunku:

```
// a i b to długości boków prostokąta pobrane od użytkownika
// warunek będzie spełniony, gdy obie zmienne
// będą większe od zero
if (a > 0 && b > 0) {
  System.out.println("Pole prostokąta wynosi " + a * b);
} else {
  System.out.println("Nieprawidłowe dane.");
}
```

✓ Zamiast operatora `&&` możemy w tym przykładzie użyć operator `||` (OR), tzn. zamiast obliczać pole prostokąta, gdy oba boki są poprawne, wypiszemy na ekran informację, że dane są nieprawidłowe, gdy jeden z boków ma błędną wartość:

```
// używamy operatora || - warunek będzie spełniony,
// gdy zmienna a będzie mniejsza bądź równa 0 lub
// zmienna b będzie mniejsza bądź równa 0
if (a <= 0 || b <= 0) {
  System.out.println("Nieprawidłowe dane.");
} else {
  System.out.println("Pole wynosi " + a * b);
}
```

✓ Operator logiczny `!` (wykrzyknik) to jednoargumentowy operator, logiczne zaprzeczenie (NOT). Działa na argumencie o wartości `true` bądź `false`, i zwraca wartość przeciwną: dla argumentu `true` zwraca `false`, a dla `false` zwraca `true`.

✓ Operatora logicznego `!` możemy użyć w celu odwrócenia wartości w instrukcji `if`. W poniższym przykładzie komunikat `"Sloneczna pogoda."` zostanie wyświetlony, gdy wartość `czyPadaDeszcz` będzie równa `false`, ponieważ zaprzeczenie `czyPadaDeszcz` będzie miało wartość `true`:

```
boolean czyPadaDeszcz = true;

// !czyPadaDeszcz -> !true -> false
// wyświetony więc zostanie komunikat "Zostaje w domu!"
if (!czyPadaDeszcz) {
  System.out.println("Sloneczna pogoda.");
} else {
  System.out.println("Zostaje w domu!");
}
```

✓ Operatora logicznego `!` możemy używać także do zaprzeczania bardziej skomplikowanych wyrażeń:

```
// jeżeli nie jest prawdą, że zarówno a, jak i b,
// są większe od zero, to znaczy, że co najmniej jedna z
// tych zmiennych ma błędną wartość
if (!(a > 0 && b > 0)) {
  System.out.println("Niepoprawne dane.");
} else {
  System.out.println("Pola prostokąta = " + a * b);
}
```

✓ Tablica prawdy to tabela informująca, jaką wartość ma wyrażenie, w którym wykorzystywany jest operator z podanymi mu argumentami.

✓ Tabela prawdy możliwych wyników zastosowania operatorów `&&`, `||`, oraz `!`:

x	y	x || y	x && y	!x
false	false	false	false	true
true	false	true	false	false
false	true	true	false	true
true	true	true	true	false

✓ Przykład dla wartości `a = 0` i `b = 100`:

a > 0	b >= 100	a > 0 || b >= 100	a > 0 && b >= 100	!(a > 0)	!(b >= 100)
false	true	true	false	true	false

- ✓ Operator `&&` ma wyższy priorytet, niż operator `||` – poniższy kod:

```
x > 0 && y > 0 || z > 0
```

jest równoznaczny z poniższym zapisem, który wykorzystuje nawiasy w celu zwiększenia czytelności:

```
(x > 0 && y > 0) || z > 0
```

Możemy zmienić kolejność porównań w tym wyrażeniu stosując nawiasy:

```
x > 0 && (y > 0 || z > 0)
```

- ✓ Jeżeli mamy kilka wyrażeń w warunku i korzystamy z różnych operatorów warunkowych `&&` i `||`, to warto stosować nawiasy aby zwiększyć czytelność kodu.
- ✓ Trójargumentowy operator warunkowy to skrócona forma prostych instrukcji warunkowych – jego składnia jest następująca:

```
wyrażenieLogiczne ? wyrażenieGdyPrawda : wyrażenieGdyFałsz
```

- ✓ W poniższym przykładzie przypiszemy do zmiennej `wartoscAbsolutna` wartość `x`, jeżeli `x` jest liczbą dodatnią. W przeciwnym razie, przypiszemy do zmiennej `wartoscAbsolutna` przeciwieństwo wartości zmiennej `x`:

```
int x = 5;
int wartoscAbsolutna;

wartoscAbsolutna = x > 0 ? x : -x;
```

- ✓ Pisząc złożone instrukcje warunkowe, często jesteśmy w stanie odpowiedzieć na pytanie, czy całe wyrażenie ma szansę być prawdą jeszcze zanim obliczymy każde z wyrażeń, które na ten warunek się składa:

```
if (a <= 0 || b <= 0) {
  System.out.println("Nieprawidłowe dane.");
} else {
  System.out.println("Pole wynosi " + a * b);
}
```

- ✓ W tej instrukcji warunkowej wystarczy, że pierwsze wyrażenie, `a <= 0` będzie prawdą, aby całe wyrażenie `a <= 0 || b <= 0` miało wartość `true`. W takim przypadku obliczanie wartości wyrażenia `b <= 0` nie ma sensu, bo i tak wiemy już, że niezależnie od wartości wyrażenia `b <= 0`, całe wyrażenie będzie miało wartość `true` (ponieważ operatorowi `||` wystarczy jeden argument o wartości `true`, aby całe wyrażenie miało wartość `true`).

- ✓ Dzięki funkcjonalności zwanej *short-circuit evaluation* nasz program nie będzie zawsze sprawdzał wszystkich warunków, jeżeli jest w stanie wcześniej jednoznacznie wyznaczyć końcową wartość danego wyrażenia. Nie musimy nic robić, aby z tej funkcjonalności korzystać – jest ona po prostu automatycznie stosowana przez Maszynę Wirtualną Java, która wykonuje kod naszego programu. Funkcjonalność ta działa zarówno dla operatora `||`, jak i operatora `&&`.

7.8.3. Porównywanie stringów

- ✓ Wartości typu `String` należy porównywać za pomocą metody `equals` typu `String` zamiast operatora `==`:

```
// zwróci false - należy użyć equals
"pass123!" == podaneHaslo

// ok - zwróci true jeżeli podaneHaslo zawiera pass123!
"pass123!".equals(podaneHaslo)
```

- ✓ Powodem jest to, że zmienne typów złożonych, takich jak typ `String`, to referencje (wskaźniki) odnoszące się do wartości typu `String`, która znajduje się gdzieś w pamięci komputera.
- ✓ **Operator porównania `==` w przypadku wartości typów złożonych nie odpowiada na pytanie: *„Czy te wartości mają identyczny stan?”* lecz na pytanie: *„Czy te wartości są tym samym obiektem w pamięci?”*.**
- ✓ Dlatego należy stosować metodę `equals`, która zwraca `true`, jeżeli przekazany do niej jako argument ciąg znaków jest taki sam, jak wartość, na rzecz której tę metodę wywołaliśmy. W przeciwnym razie zwracana jest wartość `false`.
- ✓ Metoda `equals` rozróżnia małe i wielkie litery:

```
// małe i wielkie litery są traktowane
// przez equals jako inne znaki
"pass123!".equals("PASS123!") // zwróci false
```

- ✓ Aby porównać dwa łańcuchy tekstowe traktując małe i wielkie litery jako takie same można skorzystać z metody `equalsIgnoreCase`:

```
"pass123!".equalsIgnoreCase("PASS123!") // zwróci true
```

7.8.4. Bloki kodu i zakres zmiennych

- ✓ Blok kodu to fragment zawarty w nawiasach klamrowych.
- ✓ Zmienna zdefiniowana np. w bloku instrukcji `if` będzie niedostępna, gdy wyjdziemy poza zakres tego bloku. Poniższy kod się nie skompiluje, ponieważ zmienna `wynik` nie istnieje poza blokiem `if`, w którym została zdefiniowana:

```
int licznik = 9;
int mianownik = 3;

if (mianownik != 0) {
  double wynik = licznik / mianownik;
}

// poniższa linia spowoduje błąd kompilacji
// po wyjściu z bloku kodu skojarzonego z instrukcją if
// zmienna wynik przestała istnieć
System.out.println("Wynik = " + wynik); // błąd kompilacji
```

- ✓ Zakres widoczności i „życia” zmiennych to po angielsku *scope*.

7.8.5. Typ boolean

- ✓ Zmienne typu `boolean` mogą przechowywać tylko jedną z dwóch możliwych wartości – `true` bądź `false`.
- ✓ Zmienne typu `boolean` przydają się do przechowywania informacji typu prawda/fałsz, np. `czyZalogowany`, `czyLiczbaParzysta`, `czyPokazacMenu`.
- ✓ Wyrażenie, w którym użyty jest operator relacyjny, ma wartość `true` lub `false`, czyli takie same wartości, jakie mogą przechowywać zmienne typu `boolean`, dlatego możemy przypisać do zmiennej typu `boolean` wynik takiego wyrażenia:

```
int liczba = 5;

// przypisujemy do zmiennej wynik porównania
boolean czyParzysta = liczba % 2 == 0;
```

- ✓ Operatora logicznego zaprzeczenia `!` możemy użyć, by zamienić wartość zmiennej typu `boolean` na przeciwną wartość:

```
boolean czyPadaDeszcz = false;

// zmiennej czyPadaDeszcz zostanie przypisana wartosc true
czyPadaDeszcz = !czyPadaDeszcz;
```

✓ Zmienna typu `boolean` może być użyta jako warunek w instrukcji warunkowej:

```
boolean czyPadaDeszcz = false;

// moglibyśmy napisać if (czyPadaDeszcz == true),
// ale jest to nadmiarowe i nigdy w ten sposób nie
// zapisuje się warunków z użyciem zmiennych typu boolean
if (czyPadaDeszcz) {
  System.out.println("Weź parasol!");
} else {
  System.out.println("Zostaw parasol w domu.");
}
```

7.8.6. Instrukcja i wyrażenie switch

✓ Instrukcja `switch` służy do porównania wartości wyrażenia z wylistowanymi stałymi wartościami – wykonane zostaną te instrukcje, które są skojarzone z dopasowaną wartością.

```
switch (operacja) {
  case "*":
    wynik = liczba1 * liczba2;
    break;
  case "+":
    wynik = liczba1 + liczba2;
    break;
  case "-":
    wynik = liczba1 - liczba2;
    break;
  default:
    System.out.println("Nieznana operacja.");
}
```

✓ Zadaniem słowa kluczowego `break` jest przerwanie wykonania kolejnych, następujących po nim operacji. W poniższym przykładzie brakuje użycia `break`:

```
switch (dzienTygodnia) {
  case 1: System.out.println("Poniedziałek.");
  case 2: System.out.println("Wtorek.");
  case 3: System.out.println("Środa.");
  case 4: System.out.println("Czwartek.");
  case 5: System.out.println("Piątek.");
  case 6: System.out.println("Sobota.");
  case 7: System.out.println("Niedziela.");
  default: System.out.println("Nieznany dzień tygodnia.");
}
```

Jeżeli zmienna `dzienTygodnia` miałaby wartość `5`, to na ekranie zobaczylibyśmy:

```
Piątek.
Sobota.
Niedziela.
Nieznany dzień tygodnia.
```

- ✓ Sekcja `default` jest opcjonalna. Skojarzone z nią instrukcje są wykonywane, gdy wartość żadnej sekcji `case` nie zostanie dopasowana do porównywanej wartości.
- ✓ Czasem chcemy wykonać takie same instrukcje po dopasowaniu do kilku różnych wartości. Możemy to osiągnąć zapisując kilka sekcji `case` jedna po drugiej:

```
switch (dzienTygodnia) {
  case 1: case 2: case 3: case 4: case 5:
    System.out.println("Dzień roboczy.");
    break;
  case 6: case 7:
    System.out.println("Weekend.");
    break;
  default:
    System.out.println("Nieznany dzień tygodnia.");
}
```

- ✓ Do niedawna instrukcji oraz wyrażenia `switch` można było używać tylko z kilkoma typami:
  - `byte` i `Byte`, `short` i `Short`, `int` i `Integer`
  - `char` i `Character`, `String`
  - `enum`
- ✓ Jednakże, od wersji Java 21 możemy korzystać w instrukcji i wyrażeniu `switch` z dowolnych typów złożonych. To bardziej zaawansowane zagadnienie będzie omówione w kolejnym tomie tej książki.
- ✓ Typy `long`, `Long`, `double`, `Double`, `float`, oraz `Float`, nie są wspierane przez instrukcję i wyrażenie `switch`.
- ✓ Instrukcja `switch` ma dwie możliwe składnie – od 14 wersji języka Java możemy zapisać przykład z operacją do wykonania w skrócony sposób:

```
switch (operacja) {
  case "*" -> wynik = liczba1 * liczba2;
  case "+" -> wynik = liczba1 * liczba2;
  case "-" -> wynik = liczba1 * liczba2;
  default -> System.out.println("Nieznana operacja.");
}
```

- ✓ Ta wersja instrukcji `switch` korzysta z nowej składni do wskazywania instrukcji związanej z daną sekcją `case`:

```
case wartość ->
```

- ✓ Ponadto, nie musimy już sami wstawiać do sekcji `case` instrukcji `break` – zostanie ona dodana automatycznie dla naszej wygody.
- ✓ Jeżeli chcemy wykonać więcej niż jedną instrukcję w instrukcji `switch` korzystając z nowej składni, to musimy objąć je w nawiasy klamrowe:

```
case 5 -> {
  nazwaDnia = "Piątek";
  System.out.println("Jutro weekend!");
}
```

- ✓ W nowej składni instrukcji `switch` możemy także w prostszy sposób zapisać kilka wartości skojarzonych z tymi samymi instrukcjami:

```
switch (dzienTygodnia) {
  case 1, 2, 3, 4, 5 -> System.out.println("Praca.");
  case 6, 7 -> System.out.println("Weekend.");
  default -> System.out.println("Nieznany dzień tygodnia.");
}
```

- ✓ W 14 wersji poza uproszczeniem składni instrukcji `switch` do Javy zostało dodane wyrażenie `switch`. Możemy teraz korzystać ze `switch` jako sposobu na wyznaczenie pewnej wartości i bezpośrednio przypisać ją np. do zmiennej:

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  case 2 -> "Wtorek";
  case 3 -> "Środa";
  case 4 -> "Czwartek";
  case 5 -> "Piątek";
  case 6 -> "Sobota";
  case 7 -> "Niedziela";
  default -> "Nieprawidłowy numer dnia.";
};
```

✓ Z wyrażeniami `switch` związany jest nowy mechanizm: instrukcja `yield`. Służy ona do zwracania wartości z bloku `case` wyrażenia `switch` gdy chcemy wykonać więcej niż jedną instrukcję:

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  // pozostałe wartości zostały pominięte
  case 7 -> "Niedziela";
  default -> {
    System.out.println("Podano nieprawidłowy numer dnia.");
    yield "?";
  }
};
```

✓ `yield` musimy także użyć gdy korzystamy ze starej składni (gdy zastosujemy dwukropek zamiast ->):

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1: yield "Poniedziałek";
  // pozostałe wartości zostały pominięte
  case 7: yield "Niedziela";
  default: yield "Nieprawidłowy numer dnia.";
};
```

✓ Wyrażenie `switch` musi zwracać wartość bądź kończyć się rzuceniem wyjątku (wyjątki będą omówione w drugim tomie). Nie może być sytuacji, w której pewna wartość bądź wartości nie są brane pod uwagę w wyrażeniu `switch`, co mogłoby spowodować, że wyrażenie jako całość nie miałoby wartości:

```
String nazwaDnia = switch (dzienTygodnia) {
  case 1 -> "Poniedziałek";
  case 2 -> "Wtorek";
  case 3 -> "Środa";
  case 4 -> "Czwartek";
  case 5 -> "Piątek";
  case 6 -> "Sobota";
  case 7 -> "Niedziela";
}; // błąd kompilacji - dodanie sekcji default
   // naprawiłoby problem
```

7.9. Pytania

1. Spójrz na poniższe fragmenty kodu i odpowiedz na pytanie, czy są one poprawnie zapisanym kodem źródłowym Java, a jeżeli nie, to dlaczego?

```
int x = 5;

if x == 5 {
  System.out.println("x = 5");
}
```

```
int z = 5;

if (z == 5) {
  System.out.println("z = 5");
}
else System.out.println("z > 5");
else if (z < 5) {
  System.out.println("z < 5");
}
```

```
int a = 5;

if (a) {
  System.out.println("a = 5");
}
```

```
boolean b = true;

if (b) {
  System.out.println("Warunek spełniony.");
}
```

```
int y = 5;

if (y = 5) {
  System.out.println("y = 5");
}
```

```
int c = 5;
boolean czyDodatnia = c > 0;

if (czyDodatnia) {
  System.out.println("Warunek spełniony.");
}
```

2. Jaki będzie wynik uruchomienia poniższego kodu, gdy zmienna `x` będzie równa:
   a. `10`,
   b. `20`,
   c. `30`,
   d. będzie miała inną wartość.

```
switch (x) {
  case 10:
    System.out.println("10");
  case 20:
    System.out.println("20");
    break;
  case 30:
    System.out.println("30");
  default:
    System.out.println("Inna wartość.");
}
```

3. Jaki będzie wynik działania poniższego fragmentu kodu?

```
double liczba = 50;

switch (liczba) {
  case 0 -> System.out.println("0");
  case 50 -> System.out.println("50");
  case 100 -> System.out.println("100");
  default -> System.out.println("Inna wartość.");
}
```

4. Czy poniższa instrukcja `if` oraz użycie trójargumentowego operatora warunkowego są sobie równoważne?

```
int z = 5;
int wartoscAbsolutna;

if (z > 0) {
  wartoscAbsolutna = z;
} else {
  wartoscAbsolutna = -z;
}

int wartoscAbsolutna2 = z > 0 ? -z : z;
```

5. Jak porównać ze sobą dwie wartości typu `String`?

6. W jaki sposób można skrócić poniższy kod?

```
int x = 5;
boolean czyDodatnia;

if (x > 0) {
  czyDodatnia = true;
} else {
  czyDodatnia = false;
}
```

7. Jaka będzie wartość zmiennej `pewnaZmienna`?

```
boolean pewnaZmienna;

pewnaZmienna = !pewnaZmienna;
```

8. Czy operatora = można używać do porównywania wartości?
9. Jaka będzie wartość poniższego wyrażenia, jeżeli `x = -5`, `y = -10`, `z = 20` ?

```
x > 0 && y > 0 || z > 0
```

10. Jakie wartości mogą mieć zmienne typu `boolean`?
11. Jeżeli mamy warunek `x > 0 && y > 0`, to czy wartość wyrażenia `y > 0` będzie zawsze obliczana? Jeśli nie, to dlaczego i w jakim przypadku?
12. Zapisz poniższą instrukcję `switch` za pomocą nowej składni instrukcji `switch`, wprowadzonej do języka Java w wersji 14:

```
String miesiac = "czerwiec";

switch (miesiac) {
  case "grudzień": case "styczeń": case "luty":
    System.out.println("Zima.");
    break;
  case "marzec": case "kwiecień": case "maj":
    System.out.println("Wiosna.");
    break;
  case "czerwiec": case "lipiec": case "sierpień":
    System.out.println("Lato.");
    System.out.println("Hurra!");
    break;
  case "wrzesień": case "październik": case "listopad":
    System.out.println("Jesień.");
    break;
  default:
    System.out.println("Nieznany miesiąc.");
}
```

13. Jaki będzie wynik działania poniższego fragmentu kodu?

```
int x;
x = pobierzLiczbe();

if (x >= 0) {
  int poleKwadratu = x * x;
}

System.out.println("Pole kwadratu wynosi: " + poleKwadratu);
```

14. Dla jakich wartości argumentów operatory `&&` oraz `||` zwracają `true`? A dla jakiego argumentu zwraca wartość `true` operator `!` (logiczne zaprzeczenie)?
15. Który z operatorów ma wyższy priorytet: `||` czy `&&` ?
16. Co zostanie wypisane w wyniku uruchomienia poniższego programu?

```
int x;
x = pobierzLiczbe();

if (x < 0)
  System.out.println("x jest mniejsze od 0");
  x = -x;

System.out.println(
    "Wartość absolutna pobranej liczby to: " + x
);
```

17. Co zostanie wypisane i dlaczego?

```
String komunikat = "Będzie padać";

if (komunikat.equals("będzie padać")) {
  System.out.println("Weź parasol!");
} else {
  System.out.println("Słoneczna pogoda.");
}
```

18. Jaka wartość zostanie przypisana do zmiennej `pogoda`?

```
int miesiac = 5;
String pogoda = switch (miesiac) {
  case 12, 1, 2 -> "zimno";
  case 3, 4, 5 -> "ciepło";
  case 6, 7, 8 -> "gorąco";
  case 9, 10, 11 -> "mokro";
};
```

19. Jaki będzie wynik działania poniższego fragmentu kodu?

```
int x;
int poleKwadratu;

x = pobierzLiczbe();

if (x > 0) {
  System.out.println("x jest większe od zero.");
  poleKwadratu = x * x;
}

System.out.println("Pole kwadratu wynosi: " + poleKwadratu);
```

20. Czym różni się instrukcja `switch` od wyrażenia `switch`?
21. Jaka wartość zostanie przypisana do zmiennej `rodzajDnia`?

```
int dzienTygodnia = 1;

String rodzajDnia = switch (dzienTygodnia) {
  case 6, 7 -> "Weekend.";
  default -> {
    System.out.println("Niestety - praca!");
    "Dzień roboczy.";
  }
};
```

7.10. Zadania

7.10.1. Wypisz największą z czterech liczb

Napisz program, który pobierze od użytkownika cztery liczby i wypisze największą z nich. Przygotuj dwie wersje tego programu – w pierwszej użyj instrukcji warunkowej `if` do wyznaczenia największej liczby, a w drugiej kilkukrotnie skorzystaj z metody `Math.max`, która przyjmuje dwa argumenty i zwraca większy z nich.

7.10.2. Sprawdź imię

Napisz program, który pobierze od użytkownika jego imię i odpowie na pytanie, czy jego imię jest takie samo, jak Twoje (załóżmy, że użytkownik podaje imię bez polskich znaków). Pamiętaj, aby porównać wartości typu `String` w odpowiedni sposób.

7.10.3. Czy osoba jest pełnoletnia?

Napisz program, który pobiera wiek od użytkownika. Zapisz w zmiennej typu `boolean` informację, czy użytkownik jest pełnoletni, czy nie. Na podstawie wyznaczonej wartości zmiennej `boolean` wypisz na ekran informację, czy osoba jest pełnoletnia, czy nie.

7.10.4. Malejący lub rosnący ciąg trzech liczb

Napisz program, w którym pobierzesz od użytkownika trzy liczby. Jeżeli tworzą one rosnący lub malejący ciąg w kolejności, w której zostały podane od użytkownika, to wypisz na ekran informację „ten ciąg liczb jest rosnący” lub, odpowiednio, „ten ciąg liczb jest malejący”. Jeżeli ciąg nie jest ani rosnący, ani malejący, to program powinien także wypisać taką informację. Dla przykładu, gdy użytkownik poda liczby `5`, `10`, `15`, to program powinien wypisać na ekran informację, że ciąg jest rosnący.

7.10.5. Liczenie pola figury

Napisz program, który pobierze od użytkownika nazwę figury: koło, prostokąt, bądź kwadrat (zakładamy, że użytkownik poda nazwę bez polskich liter). Następnie, program powinien pobrać od użytkownika: promień, jeżeli użytkownik wybrał koło, bok dla kwadratu, lub dwa boki dla prostokąta.

Program powinien wyliczyć pole danej figury na podstawie pobranych wartości i wypisać wynik na ekran. Jeżeli wczytane wartości będą nieprawidłowe (nieznany rodzaj figury lub ujemne wartości promienia/boków), to program powinien wypisać stosowny komunikat.

7.10.6. Sprawdzanie mnożenia

Napisz program, który wylosuje dwie liczby z przedziału od `1` do `20` (włącznie). Wypisz je na ekran i zapytaj użytkownika, ile wynosi iloczyn (mnożenie) tych dwóch liczb. Jeżeli użytkownik poda poprawną odpowiedź, wypisz stosowny komunikat. W przeciwnym razie poinformuj użytkownika ile wynosi ten iloczyn.

Aby wylosować liczbę z przedziału od `1` do `20` wykorzystaj ten fragment kodu:

```
int liczba1 = ThreadLocalRandom.current().nextInt(20) + 1;
```

Na początku swojego programu będziesz musiał(a) dodać stosowną instrukcję `import`, aby móc używać klasy `ThreadLocalRandom` z Biblioteki Standardowej Java:

```
import java.util.concurrent.ThreadLocalRandom;
```

7.10.7. Zamiana nazwy snake_case na Camel Case

Napisz program, który pobierze od użytkownika jedno słowo – nazwę składającą się z dwóch słów zapisaną w standardzie snake_case (słowa są zapisane małymi literami i połączone są ze sobą znakiem podkreślenia). Zamień tę nazwę, aby była zapisana w standardzie Camel Case (wszystkie litery są małe poza każdą pierwszą literą zaczynając od drugiego słowa) i wypisz wynik na ekran.

Będziesz musiał(a) skorzystać z metody `indexOf` typu `String`. Sprawdź w dokumentacji typu `String` jak działa ta metoda. Jeżeli użytkownik poda nazwę, która nie zawiera podkreślenia, program powinien wypisać komunikat, że spodziewał się nazwy w standardzie snake_case składającej się z dwóch słów, oddzielonych od siebie znakiem podkreślenia. Przykładowe uruchomienie:

```
Podaj nazwę składającą się z dwóch słów zapisaną w standardzie snake_case:
moja_nazwa
Ta nazwa zapisana w standardzie Camel Case to mojaNazwa
```

```
Podaj nazwę składającą się z dwóch słów zapisaną w standardzie snake_case: nazwa_
Podana nazwa nie składa się z dwóch słów.
```

```
Podaj nazwę składającą się z dwóch słów zapisaną w standardzie snake_case: nazwa
W podanej nazwie nie ma znaku podkreślenia.
```

7.10.8. Magiczna kula nr 8

Napisz program, w którym użytkownik zada pytanie, a program wypisze jedną z ośmiu losowych odpowiedzi:

„Tak.”, „Być może.”, „Raczej nie.”, „Nie.”, „Nie wiem.”, „Nie liczyłbym na to.”, „Lepiej, żebyś nie wiedział(a).”, „W tej chwili Ci nie powiem, spróbuj później.”.

Aby wylosować liczbę, skorzystaj z podobnego kodu, jak w zadaniu „7.10.6 Sprawdzanie mnożenia”, z tym, że jako argument do metody `nextInt` podaj `8`.

Aby wczytać całą linię (zamiast pojedynczego słowa) skorzystaj z metody `nextLine` obiektu `Scanner` (to ten sam obiekt, z którego korzystamy wczytując od użytkownika liczby i pojedyncze słowa):

```
public static String pobierzLinieTekstu() {
  return new Scanner(System.in).nextLine();
}
```

Przykładowe uruchomienie programu:

```
Co chcesz wiedzieć?
Czy jutro będzie padać?
Pytasz „Czy jutro będzie padać?”... W tej chwili Ci nie powiem, spróbuj później.
```