JAVA
OD PODSTAW

Tom 1 – od pierwszego programu do pisania metod

Odpowiedzi na pytania i rozwiązania do zadań

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

Przemysław Kruglej

Odpowiedzi na pytania i rozwiązania do zadań z książki:

„Java od podstaw – Tom 1 (wydanie 04-2024)**”**

autorstwa Przemysława Krugleja.

Wersja dokumentu: 16-05-2024

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 książki „Java od podstaw”:

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

SPIS TREŚCI

WSTĘP

Witaj!

Ten dokument zawiera odpowiedzi na pytania i rozwiązania do zadań z książki „Java od podstaw – Tom 1” mojego autorstwa. Mam nadzieję, że przygotowane przeze mnie pytania oraz zadania pomogą Ci podczas nauki języka Java. Jeżeli będziesz miał(a) jakieś uwagi bądź pytania, możesz skontaktować się ze mną mailowo lub odwiedzić mój kanał Discord – więcej informacji jak do niego dołączyć znajdziesz w jednym z kolejnych podrozdziałów.

Mój adres e-mail: przemyslaw.kruglej@gmail.com

Przemysław Kruglej

O książce „Java od podstaw”

Jeżeli rozpoczynasz swoją przygodę z językiem Java, to moja książka „Java od podstaw” jest dla Ciebie.

Napisałem ją z myślą o osobach początkujących, które nie miały do tej pory styczności z programowaniem lub miały w niewielkim stopniu, a chciałyby spróbować programowania w Javie i, być może, myślą o sprawdzeniu swoich sił w zawodzie programisty. Wiedza zawarta w mojej książce jest solidnym punktem startowym, po którym będzie można kontynuować naukę.

Jeżeli chcesz nauczyć się programowania aby mieć przyjemność z pisania własnych programów, to po przeczytaniu „Javy od podstaw” będziesz miał(a) solidne ku temu podstawy. Jeżeli myślisz o pracy jako programista, to przygotuj się na długą i interesującą przygodę.

Moim celem jest nie tylko nauczenie Cię języka Java, ale także przedstawienie najlepszych praktyk związanych z programowaniem, których znajomość dobrze świadczy podczas ewaluacji potencjalnego kandydata na stanowisko programisty.

Książka dostępna jest do zakupu na mojej oficjalnej aukcji Allegro:

https://allegro.pl/oferta/java-od-podstaw-tom-1-przemyslaw-kruglej-15656583058

Zależy mi, aby zarówno moje książki, jak i inne materiały im towarzyszące, były najlepszej jakości, jaką mogę dostarczyć. Dołożyłem wszelkich starań, aby treść była klarowna, przystępna, i przydatna. Jeżeli znajdziesz w moich tekstach jakiekolwiek błędy, chociażby drobne literówki, lub cokolwiek będzie niezrozumiale wytłumaczone lub uważasz, że pominąłem coś istotnego – proszę, daj mi znać. Naniosę wszystkie poprawki, natomiast niezrozumiałe bądź niewystarczające opisy zagadnień postaram się poprawić. Z góry dziękuję! Możesz do mnie napisać na poniższy adres e-mail lub skontaktować się ze mną na moim kanale Discord, o którym zaraz opowiem Ci więcej.

Mój adres e-mail: przemyslaw.kruglej@gmail.com

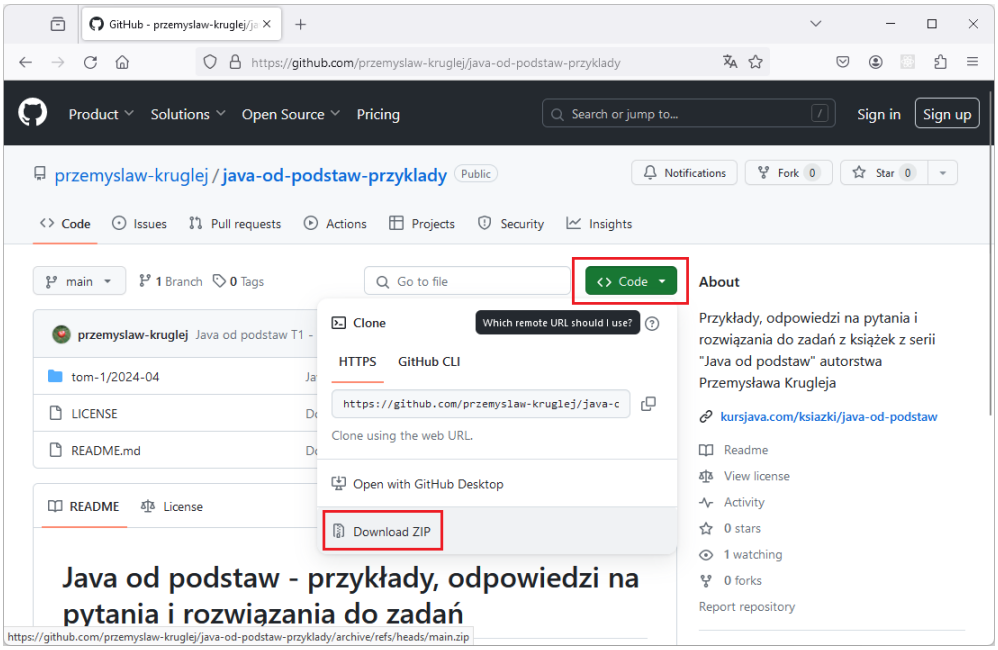
Kod źródłowy oraz odpowiedzi na pytania i zadania

Kody źródłowe ze wszystkimi przykładami, które zawarte są w książce „Java od podstaw”, znajdziesz w poniższym repozytorium na Githubie:

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

Przykłady znajdują się w katalogu „tom-1” w podkatalogu „2024-04”, gdzie są podzielone na poszczególne rozdziały. Aby je ściągnąć, nie jest wymagane konto na stronie Github. Całe repozytorium można wygodnie pobrać w postaci archiwum *.zip* klikając na zielony przycisk „Code”, a następnie na „Download ZIP” w menu, które się pojawi, co prezentuje Rysunek 1.

Ten dokument oraz kody źródłowe rozwiązań do zadań znajdziesz w katalogu „tom-1/2024-04” w podkatalogu „Pytania_i_zadania_rozwiazania”.



Rysunek 1 – Zaznaczone przyciski Code oraz Download ZIP, pozwalające na pobranie zawartości repozytorium z przykładami do książki

Formatowanie używane w tym dokumencie

W tym dokumencie będę stosował następujące formatowanie:

- Techniczne wyrażenia, takie jak nazwy zmiennych oraz typów, będą w tekście formatowane w następujący sposób: `String`.
- Słowa kluczowe języka Java, liczby, znaki, oraz stringi (łańcuchy tekstowe), będą wyglądały jak poniżej:
  - `public`, `int`, `return`
  - `100`, `10.0f`, `3.14`
  - `'a'`, `'Z'`
  - `"Witaj"`[1], `"Wynik wynosi:"`
- Klawisze klawiatury będą formatowane następująco: ENTER
- Kod źródłowy będzie prezentowany w poniższy sposób. Jeżeli ten kod będzie zawarty w repozytorium z przykładami, to na początku dodam informację o ścieżce do pliku, w którym się on znajduje. Dodatkowo, jeżeli będę

[1] Jeżeli w tekście będzie informacja, że pewna zmienna ma wartość, dla przykładu, `"Witaj"`, to będzie to oznaczało, że zawiera ona pięć znaków: W, i, t, a, oraz j. Znaki cudzysłowu mają na celu jedynie uwypuklenie, że chodzi o łańcuch tekstowy, czyli string (obiekt typu `String` w języku Java). Cudzysłowy nie stanowią części wartości przechowywanej w danym stringu.

chciał, abyś zwrócił(a) uwagę na pewien fragment kodu, to zostanie on podświetlony. Jeżeli będę omawiał kod źródłowy, to za pomocą komentarzy z kolejnymi numerami, na przykład `// 1`, będę zaznaczał istotne linie, a następnie w tekście będę się do nich odnosił za pomocą odpowiedniego numeru zawartego w nawiasach, np. `(1)`:

przykłady/wstęp/Powitanie.java

```
import java.util.Scanner;

public class Powitanie {
  public static void main(String[] args) {
    System.out.print("Podaj swoje imię: ");
    String imie = pobierzSlowo();

    System.out.printf("Witaj %s!\n", imie); // 1
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

- Komendy wywoływane z linii poleceń będą prezentowane w poniższej ramce. Używana komenda będzie dodatkowo zaznaczona **wytłuszczoną** czcionką i będzie poprzedzona znakiem większości >. Dane wprowadzane z klawiatury przez użytkownika będą natomiast miały dodatkowe tło:

```
> java Powitanie
Podaj imię: Przemek
Witaj Przemek!
```

- Dodatkowe informacje będą prezentowane w następujący sposób:

Dodatkowe informacje

Pamiętaj, aby nie dzielić przez zero!

- Dobre praktyki będą zawarte w poniższej ramce:

Dobre praktyki

Stosowanie się do konwencji i przyjętych zasad pisania kodu źródłowego zwiększa jego czytelność.

- Jeżeli jakaś funkcjonalność języka Java została do niego dodana w ostatnich latach, to zaznaczę to odpowiednim numerem wersji w nawiasach kwadratowych: [Java 14].

O autorze

Jestem programistą z zamiłowania – zainteresowałem się programowaniem mając 14 lat i od tej pory stało się ono moją pasją. Jako programista pracowałem na kilkunastu projektach przez niespełna 15 lat korzystając z różnych technologii i zdobywając cenne doświadczenie.

Lubię uczyć oraz dzielić się wiedzą, dlatego kilka lat temu zacząłem tworzyć kursy przeznaczone dla osób, które chcą się nauczyć programowania od podstaw.

Chciałbym dzięki moim książkom, kursom, oraz artykułom ułatwić rozpoczęcie przygody z programowaniem osobom, które nie miały z nim do tej pory styczności.

Staram się, aby wszystkie tworzone przeze mnie treści były najwyższej jakości i abyś, Droga Czytelniczko / Drogi Czytelniku, jak najwięcej się dzięki nim nauczył(a). Dlatego, jeżeli cokolwiek będzie dla Ciebie niejasne bądź niezrozumiałe, nie wahaj się do mnie napisać na adres:

przemyslaw.kruglej@gmail.com

Będę bardzo wdzięczny za wszelkie uwagi i pomysły jak usprawnić ten dokument.

Dodatkowe materiały

Na mojej stronie – https://kursjava.com – znajdziesz wiele darmowych kursów od podstaw, a także artykuły mające na celu przybliżenie Ci jak wygląda praca jako programista oraz co trzeba umieć, aby przebranżowić się z innego zawodu. Znajdziesz tam między innymi:

- Kurs *„Java od podstaw”*, na którym oparta jest część pierwszego tomu mojej książki „Java od podstaw”: https://kursjava.com/wstep-do-kursu
- Kurs *„Podstawy Maven”*: https://kursjava.com/podstawy-maven
- Kurs *„Tworzenie gier w języku Java”*: https://kursjava.com/tworzenie-gier
- Kurs *„IntelliJ IDEA w akcji”*: https://kursjava.com/intellij-idea-w-akcji
- Kurs *„Podstawy linii poleceń”*: https://kursjava.com/podstawy-linii-polecen
- Zagadki Java: https://kursjava.com/zagadki-java
- Artykuły z serii *„Przebranżowienie i praca programisty”*:
    - Co trzeba umieć, aby zostać programistą?
        - Część 1 – Najistotniejsze technologie i narzędzia: https://kursjava.com/przebranzowienie-i-praca-programisty/co-trzeba-umiec-aby-zostac-programista-czesc-1

        - Część 2 – Dodatkowe zagadnienia:
          https://kursjava.com/przebranzowienie-i-praca-programisty/co-trzeba-umiec-aby-zostac-programista-czesc-2
        - Część 3 – Technologie i narzędzia, które warto znać:
          https://kursjava.com/przebranzowienie-i-praca-programisty/co-trzeba-umiec-aby-zostac-programista-czesc-3
    - Wywiad z osobą, która przebranżowiła się na programistę:
      https://kursjava.com/przebranzowienie-i-praca-programisty/wywiad-z-osoba-ktora-przebranzowila-sie-na-programiste
    - Jak zostać programistą?
      https://kursjava.com/przebranzowienie-i-praca-programisty/jak-zostac-programistaOpis

Na początku nauki programowania w szczególności przydatne mogą się okazać kursy *„Podstawy linii poleceń”* oraz *„IntelliJ IDEA”* w akcji.

Moje inne książki

Napisałem do tej pory dwie inne książki dla osób, którą uczą się programować, i chciałyby zostać zawodowymi programistami:

- *„Java w przykładach”* – książka przeznaczona jest dla osób, które uczą się programowania od podstaw w języku Java – zawiera 70 przemyślanych, dobrze opisanych zadań, których rozwiązanie pomoże w utrwaleniu podstawowych zagadnienia związanych z programowaniem w Javie.

  Do każdego z ćwiczeń przygotowałem rozwiązanie, które opisuję i poddaję dokładnej analizie. Zadania podzielone są tematycznie na osiem rozdziałów. Zaczynamy od podstaw, kończąc na klasach i zadaniach algorytmicznych.

  Ta książka jest wydrukowana w kolorze, aby dodatkowo ułatwić zrozumienie omawianych kodów źródłowych. Więcej o tej książce, wraz z informacją, gdzie można ją kupić, znajdziesz na dedykowanej stronie:

  https://kursjava.com/ksiazki/java-w-przykladach

- *„Craftsmanship of Software Development”* – jest to zbiór moich myśli, nagromadzonych przez 15 lat zawodowej pracy, na temat tego jak można być lepszym programistą i współpracownikiem.

  Nie musisz kupować tej książki, jeżeli jesteś zainteresowany(-a) jej treścią, ponieważ jest ona oparta na moim dokumencie pod tym samym tytułem, który jest dostępny za darmo w Internecie na stronie:

  https://craftsmanshipof.software

  Więcej o tej książce, wraz z informacją, gdzie można ją kupić, znajdziesz na dedykowanej stronie:

  https://kursjava.com/ksiazki/craftsmanship-of-software-development

Mam w planach napisanie kolejnych książek, w szczególności kolejnych tomów *„Javy od podstaw”*, w których poruszę bardziej zaawansowane tematy, takie jak klasy i dziedziczenie.

Jeżeli jesteś zainteresowany(-a) moimi publikacjami, to polecam zaglądać na mój blog na stronie:

https://kursjava.com/blog

Dowiesz się z niego czy napisałem kolejne książki. Możesz się też zapisać do newslettera na mojej stronie. Lista wszystkich wydanych przeze mnie do tej pory książek jest także dostępna pod adresem:

https://kursjava.com/ksiazki

Kanał Discord

Discord to strona, na której można za darmo prowadzić rozmowy w formie chatu. Założyłem na niej kanał, na który serdecznie zapraszam.

Jeżeli masz pytania odnośnie programowania lub pracy programisty, to możesz je tam zadać – z chęcią pomogę/odpowiem! Jest to także dobre miejsce, aby porozmawiać o tym dokumencie lub moich książkach, jeżeli będziesz miał(a) do nich jakieś uwagi bądź pytania.

Informacje o moim kanale Discord „Kurs Java” oraz jak do niego dołączyć znajdziesz na mojej stronie:

https://kursjava.com/discord

INSTALACJA JAVA & KONFIGURACJA ŚRODOWISKA

Ten rozdział wyjątkowo nie zawiera pytań i zadań.

PIERWSZY PROGRAM

2.1. Pytania

1. Czym jest proces kompilacji?

Jest to proces, w którym program zwany *kompilatorem* zamienia kod źródłowy zrozumiały dla człowieka na kod zrozumiały dla komputera, lub na *kod pośredni*, który może zostać *zinterpretowany* przez inny program.

---

2. Jak nazywa się kompilator Java i jak się go używa?

Kompilator języka Java to program o nazwie `javac`. Aby go użyć, w linii poleceń wpisujemy jego nazwę, po której jako argument przekazujemy nazwę pliku z kodem źródłowym, który ma zostać skompilowany, np.:

```
> javac HelloWorld.java
```

---

3. Co powstaje w wyniku kompilacji kodu Java?

W wyniku kompilacji kodu Java powstaje tzw. *bytecode*, który może zostać uruchomiony w *Maszynie Wirtualnej Java* i w niej zinterpretowany.

---

4. Jak uruchomić kod Java?

Należy w linii komend uruchomić *Maszynę Wirtualną Java*, czyli program o nazwie `java`, któremu przekażemy jako argument nazwę skompilowanej klasy (bez podawania rozszerzenia `.class`), na przykład:

```
> java HelloWorld
```

---

5. Czym różni się kompilator języka Java od Maszyny Wirtualnej Java?

Kompilator ma za zadanie skompilować kod Java do bytecode'u, natomiast Maszyna Wirtualna Java ten bytecode interpretuje i wykonuje.

---

6. Jak powinien nazywać się plik z poniższym kodem Java?

```
public class Zagadka {
  public static void main(String[] args) {
    System.out.println(
        "Cegła waży kilo i pół cegły – ile waży cegła?"
    );
  }
}
```

Plik z tym kodem źródłowym powinien nazywać się `Zagadka.java`

---

7. Mając w katalogu plik o nazwie `TestowyProgram.java` z kodem źródłowym Java, czy poniższa komenda jest poprawna?

```
> javac TestowyProgram
```

Nie jest to poprawne uruchomienie programu `javac`, ponieważ korzystając z kompilatora języka Java powinniśmy jako argument przekazać nazwę pliku wraz z rozszerzeniem. Poprawne użycie:

```
> javac TestowyProgram.java
```

---

8. Mając plik o nazwie `TestowyProgram.class` ze skompilowanym kodem źródłowym Java, czy poniższa komenda jest poprawna?

```
> java TestowyProgram.class
```

Nie jest to poprawne uruchomienie Maszyny Wirtualnej Java, ponieważ chcąc przekazać jej plik z bytecodem powinniśmy podać argument bez rozszerzenia `.class`. Poprawne użycie:

```
> java TestowyProgram
```

---

9. Mając w katalogu plik o nazwie `TestowyProgram.java` z kodem źródłowym Java, czy poniższa komenda jest poprawna?

```
> java TestowyProgram.java
```

Od 11 wersji Javy możliwe jest uruchamianie programów składających się z pojedynczych plików z pominięciem manualnego procesu kompilacji przez kompilator `javac`, bezpośrednio za pomocą Maszyny Wirtualnej Java. Komenda zaprezentowana w tym pytaniu jest więc poprawna – jej wynikiem będzie skompilowanie kodu programu `TestowyProgram` i uruchomienie go w Maszynie Wirtualnej Java.

---

10. Jakie jest specjalne znaczenie metody `main`?

Jest to wyjątkowa metoda, ponieważ to od niej rozpoczyna się wykonanie każdego programu napisanego w języku Java.

---

11. Jak wypisać na ekran tekst `Witajcie!` ?

Należy skorzystać z instrukcji `System.out.println`. W nawiasach, zawarty w cudzysłowie, powinien znaleźć się komunikat do wyświetlenia, na przykład:

```
System.out.println("Witajcie!");
```

---

12. Jaki będzie efekt próby kompilacji każdego z zaprezentowanych programów?

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

Kompilacja zakończy się błędem, ponieważ na końcu drugiej linii brakuje nawiasu klamrowego `{` otwierającego ciało metody `main`.

```
public class DrugiPrzyklad {
  public static void main(String[] args) {
    System.out.println("Druga zagadka.")
  }
}
```

Kompilacja zakończy się błędem, ponieważ na końcu trzeciej linii brakuje średnika kończącego instrukcję wypisującą na ekran komunikat.

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

Kompilacja zakończy się sukcesem. Ciała metod mogą być puste.

```
public class CzwartyPrzyklad {
  public static void main(String[] args) {
    System.out.println('Czwarta zagadka.');
  }
}
```

Kompilacja zakończy się błędem, ponieważ komunikat do wyświetlenia ujęty jest w apostrofy, zamiast w cudzysłów. W języku Java *stringi*, czyli łańcuchy tekstowe, należy umieszczać w cudzysłowie. W apostrofach umieszczamy pojedyncze znaki, będące wartościami typu `char`.

```
public class PiatyPrzyklad {
  public static void main(String[] args) {
    System.out.println(”Piata zagadka.”);
  }
}
```

Kompilacja zakończy się błędem, ponieważ komunikat do wyświetlenia ujęty jest w specjalne znaki cudzysłowu, używane w edytorach dokumentów, takich jak Microsoft Word, zamiast w zwykły cudzysłów " ".

13. Które z poniższych stwierdzeń jest prawdziwe?
    a. System Windows rozumie bytecode.

Nieprawda – to Maszyna Wirtualna Java rozumie bytecode.

---

    b. Program `javac` jest potrzebny do uruchomienia skompilowanego kodu Java.

Nieprawda – program `javac`, czyli kompilator języka Java, służy do kompilacji kodu Java, a nie do jego uruchamiania.

---

    c. Stringi (łańcuchy tekstowe) powinny być zawarte w apostrofach.

Nieprawda – łańcuchy tekstowe powinny być ujęte w cudzysłów.

---

    d. Ciało metody zawarte jest pomiędzy nawiasami: ( )

Nieprawda – ciało metody powinno być zawarte w nawiasach klamrowych `{ }`

---

    e. Jeżeli kompilator napotka problemy w naszym kodzie, to nie wygeneruje pliku z rozszerzeniem `.class` ze skompilowanym kodem.

Prawda.

2.2. Zadania

2.2.1. Wypisz imię

Napisz program, który wypisze na ekran tekst `Cześć`, po którym nastąpi Twoje imię.

Rozwiązanie jest podobne do programu `HelloWorld`, czyli pierwszego programu, jaki napisaliśmy w języku Java.

Wzorując się na tamtym przykładzie, możemy utworzyć plik o nazwie:

```
WypiszImie.java
```

następnie skopiować do niego kod programu `HelloWorld`, zmienić nazwę klasy na `WypiszImie`, a na końcu zmodyfikować wypisywany na ekran komunikat:

Pytania_i_zadania_rozwiazania/Rozdzial_02/WypiszImie.java

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

Ten program po uruchomieniu wypisuje na ekran tekst:

```
Cześć Przemek
```

KOMENTARZE & FORMATOWANIE KODU

3.1. Pytania

1. Jak zapisujemy każdy rodzaj komentarza w Javie?

W Javie istnieją trzy rodzaje komentarzy:

- jednolinijkowe – zaczynają się od dwóch znaków slash `//` i kończą się wraz z końcem linii,
- wielolinijkowe – zaczynają się od znaków `/*` oraz kończą znakami `*/`.
- wielolinijkowe–dokumentacyjne – zaczynają się od znaków `/**` oraz kończą znakami `*/`.

---

2. Które komentarze mogą być zagnieżdżone?

Komentarze jednolinijkowe mogą być w sobie zagnieżdżone, w przeciwieństwie do komentarzy wielolinijkowych oraz komentarzy dokumentacyjnych.

Z drugiej strony, komentarze jednolinijkowe mogą być zawarte w komentarzach wielolinijkowych oraz na odwrót – zaprezentowane poniżej komentarze są poprawne:

```
// komentarz jednolinijkowy /* inny komentarz */
/*
komentarz wielolinijkowy
// druga linia
trzecia linia
*/
```

---

3. Co można poprawić w poniższym kodzie?

```
public class zadaniezleformatowanie {
  public static void main(String[] args) {
  // System.out.println("Witaj");
  // System.out.println("Swiecie!");
  System.out.println("Witaj Świecie!");
  }
}
```

- Nazwa klasy powinna być zapisana w standardzie *Pascal Case*, który jest podobny do *Camel Case*. W konwencji Pascal Case wszystkie słowa składające się na nazwę zaczynają się wielką literą, a pozostałe litery są małe. Nazwa tej klasy, zapisana w Pascal Case, powinna wyglądać następująco:

```
ZadanieZleFormatowanie
```

- Zakomentowany kod powinien zostać usunięty.
- Wcięcie przed instrukcją wypisująca na ekran tekst powinno być o jeden poziom większe.

Kod po poprawieniu wygląda następująco:

```
public class ZadanieZleFormatowanie {
  public static void main(String[] args) {
    System.out.println("Witaj Świecie!");
  }
}
```

---

4. Która z poniższych nazw zapisana jest w konwencji Camel Case?
    a. nazwa
    b. MojaNazwa
    c. mojanazwa
    d. Wiadomosc

Tylko pierwsza – chociaż zawiera ona tylko jedno słowo, to jest zgodna ze standardem Camel Case. Nazwa `MojaNazwa` powinna mieć pierwszą małą literę, tak samo jak nazwa `Wiadomosc`. Nazwa `mojanazwa` powinna mieć wielką pierwszą literę drugiego słowa.

---

5. Która z nazw klas jest poprawna zgodnie ze standardem nazewnictwa klas Java?
    e. HelloWorld
    f. helloworld
    g. helloWorld
    h. Helloworld

Jedyną nazwą klasy zgodną ze standardem nazewnictwa klas w języku Java jest `HelloWorld`. Nazwy klas powinny być zapisane w standardzie Pascal Case, w którym wszystkie słowa w nazwie zaczynają się wielką literą, a pozostałe litery są małe.

3.2. Zadania

3.2.1. Dopisz komentarze

Dopisz do naszego pierwszego programu, wypisującego tekst `Witaj Świecie!`, kilka komentarzy różnych typów.

Do tego programu możemy dopisać następujące komentarze:

Pytania_i_zadania_rozwiazania/Rozdzial_03/HelloWorldZKomentarzami.java

```
// poniższy program wypisuje na ekran komunikat
public class HelloWorldZKomentarzami {
  /*
    Początek metody main, która jest uruchamiana
    przez Maszynę Wirtualną Java.
  */
  public static void main(String[] args) {
    // ta instrukcja wypisuje na ekran podany tekst
    System.out.println("Witaj Świecie!");
  }
}
```

3.2.2. Brak main

W naszym pierwszym programie, który wypisuje na ekran tekst `Witaj Świecie!`, zakomentuj całą metodę `main`. Czy program się skompiluje? Czy da się go uruchomić, i jeżeli tak, to co zobaczymy na ekranie?

Ten program z zakomentowaną metodą `main` wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_03/HelloWorldBezMain.java

```
public class HelloWorldBezMain {
  /*public static void main(String[] args) {
    System.out.println("Witaj Świecie!");
  }*/
}
```

Kompilacja tego programu zakończy się bez problemu, jednakże po uruchomieniu jego działanie zakończy się następującym błędem:

```
Error: Main method not found in class HelloWorldBezMain, please define the
main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

Ten błąd jest wynikiem tego, że Maszyna Wirtualna Java próbowała wywołać metodę `main` z naszej klasy. Metoda `main` jest punktem startowym wykonywania programów napisanych w języku Java – Maszyna Wirtualna Java zgłosiła błąd, ponieważ metoda `main` nie została znaleziona.

ZMIENNE, STAŁE & TYPY PRYMITYWNE

4.1. Pytania

1. Jak definiujemy zmienne?

Zmienne definiujemy podając, kolejno, ich typ, nazwę, oraz ewentualną wartość początkową. Możemy zdefiniować więcej niż jedną zmienną na raz oddzielając ich nazwy przecinkami:

```
int bokKwadratu;
int dzienTygodnia, miesiac, rok;
char znak = 'a';
double x = 5.5, y = 10;
float promienKola = 6.48f;
String komunikat = "Będzie padać.";
```

Ponadto, od 10 wersji języka Java możemy definiować zmienne za pomocą słowa kluczowego `var` – zastępuje ono typ zmiennej, który zostanie automatycznie wyznaczony na podstawie wartości, którą zmienna zostanie zainicjalizowana:

```
var liczba = 10; // zmienna liczba będzie typu int
var tekst = "Java"; // zmienna tekst będzie typu String
```

---

2. Czy zmiennej trzeba nadać wartość początkową w momencie definicji?

Nie, ale przed użyciem takiej zmiennej należy jej przypisać wartość.

Jeżeli spróbujemy użyć zmiennej zdefiniowanej w metodzie zanim nadamy jej wartość, to program w ogóle się nie skompiluje:

```
public class UzycieNiezainicjalizowanejZmiennej {
  public static void main(String[] args) {
    int x;

    // błąd kompilacji
    // nie nadaliśmy zmiennej x jeszcze żadnej wartości
    System.out.println("Wartość x wynosi: " + x);
  }
}
```

Błąd kompilacji będzie następujący:

```
UzycieNiezainicjalizowanejZmiennej.java:7: error: variable x might not
have been initialized
    System.out.println("Wartość x wynosi: " + x);
                                              ^
1 error
```

Ta zasada odnosi się do *zmiennych lokalnych*, czyli definiowanych w metodach. *Pola klas*, które także są zmiennymi, są automatyczne inicjalizowane wartościami domyślnymi dla danego typu i można z nich korzystać bez nadania im wstępnej wartości. Pola klas są tematem drugiego tomu *„Javy od podstaw”*.

---

3. Jakie są zasady nazewnictwa zmiennych i innych obiektów w Javie?

Nazwy w języku Java, w tym nazwy: zmiennych, metod, klas, i innych obiektów, muszą spełniać trzy następujące wymagania:

- nazwa musi się zaczynać od litery, podkreślenia _ bądź znaku dolara `$`,
- kolejnymi znakami, poza tymi wymienionymi powyżej, mogą być też cyfry,
- nazwy nie mogą być takie same jak zastrzeżone słowa w języku Java (takie jak np. `class`, `public`, `void`).

---

4. Czy wielkie litery są rozróżniane w nazwach w Javie?

Tak. Dla przykładu, `pi` oraz `PI` to dwie różne nazwy w języku Java.

---

5. Jakie typy prymitywne są dostępne w Javie? Czym się różnią?

Java oferuje 8 typów prymitywnych: `char`, `byte`, `short`, `int`, `long`, `float`, `double`, oraz `boolean`. Różnią się one rodzajem danych, jakie mogą przechowywać, np. zmienne typu `boolean` mogą przechowywać wartości `true` bądź `false`, a zmienne typu `double` – wartości liczbowe z częścią ułamkową, np. `3.14`.

Ponadto, te typy różnią się także zakresem wartości, które mogą reprezentować. Wynika on z rozmiaru zmiennych danego typu, np. zmienne typu `byte` mogą przechowywać mniejszy zakres wartości, niż zmienne typu `int`, ale zajmują mniej miejsca w pamięci komputera.

6. Jak nazywamy zwykłe wartości zapisane w kodzie, takie jak `3.14`, `25`, `'a'`?

Wartości te nazywamy *literałami*. Są to wartości różnych typów zapisane bezpośrednio w kodzie źródłowym programów napisanych w języku Java.

---

7. Do czego służą stałe, jak się je definiuje i nazywa?

Stałe służą do przechowywania wartości, które nigdy nie powinny się zmienić w trakcie wykonywania programu, np. stała o nazwie `PI` powinna mieć wartość `3.14`.

Stałe definiujemy podobnie jak zmienne, ale przed typem dodajemy słowo kluczowe `final`. Stałe nazywamy używając wielkich liter i oddzielając od siebie słowa znakiem podkreślenia (jest to tzw. konwencja SCREAMING_SNAKE_CASE):

```
final int LICZBA_DNI_W_TYGODNIU = 7;
```

---

8. Co zostanie wypisane w wyniku działania poniższego programu?

```
public class WypiszX {
  public static void main(String[] args) {
    int x;
    System.out.println("x ma wartość " + x);
  }
}
```

Kod w ogóle się nie skompiluje, ponieważ nie zainicjalizowaliśmy zmiennej `x` żadną wartością. Kompilator jest to w stanie wykryć i zgłosi błąd już na etapie kompilacji:

```
WypiszX.java:4: error: variable x might not have been initialized
    System.out.println("x ma wartość " + x);
                                         ^
1 error
```

---

9. Czy poniższy kod jest poprawny?

```
int liczba = 2.5;
```

Przypisanie wartości zmiennoprzecinkowej (zawierającej część ułamkową) do zmiennej typu `int` nie jest poprawne. Zmienne tego typu mogą przechowywać liczby całkowite – powyższa linia kodu wygeneruje następujący błąd kompilacji:

```
NiepoprawnePrzypisanieWartosci.java:3: error: incompatible types: possi-
ble lossy conversion from double to int
    int liczba = 2.5;
                 ^
1 error
```

---

10. Które z poniższych nazw zmiennych są niepoprawne i dlaczego?

```
char ZNAK;
int class;
double $saldo;
int liczbaPrzedmiotow#;
int 60godzin;
```

Nieprawidłowymi nazwami są:

- `class` – jest to słowo kluczowe w języku Java i nie może być używane jako nazwa zmiennej,
- `liczbaPrzedmiotow#` – zawiera niedozwolony znak `#`,
- `60godzin` – nazwy nie mogą zaczynać się od cyfr.

Nazwa `ZNAK` jest co prawda poprawna, ale powinna zostać zapisana małymi literami – tylko stałe zapisujemy wielkimi literami. Nazwa `$saldo` jest poprawna – nazwy mogą zaczynać się od znaku dolara.

---

11. Co zostanie wypisane w wyniku działania poniższego programu?

```
public class WypiszXY {
  public static void main(String[] args) {
    int x = 10;
    int y = -5;
    System.out.println("Zmienne X i Y to: " + X + ", " + Y);
  }
}
```

Kod w ogóle się nie skompiluje – kompilator przerwie kompilację, ponieważ nie wie czym są `X` oraz `Y`. Zdefiniowane w tym programie zmienne zapisane są małymi literami – w języku Java wielkość znaków ma znaczenia.

Kompilator wyświetli następujące komunikaty błędów:

```
WypiszXY.java:5: error: cannot find symbol
    System.out.println("Zmienne X i Y to: " + X + ", " + Y);
                                              ^
  symbol:   variable X
  location: class WypiszXY
WypiszXY.java:5: error: cannot find symbol
    System.out.println("Zmienne X i Y to: " + X + ", " + Y);
                                                    ^
  symbol:   variable Y
  location: class WypiszXY
2 errors
```

---

12. Czy poniższy kod skompiluje się poprawnie?

```
char z = "Z";
System.out.println(z);
```

Nie, ponieważ do zmiennej typu `char` próbujemy przypisać string, czyli łańcuch znaków – wartość typu `String`. Chociaż złożony tylko z jednego znaku, nadal jest to łańcuch znaków, a nie pojedynczy znak. Aby kod był poprawny, do zmiennej powinien zostać przypisany znak ujęty w apostrofy `'Z'`.

---

13. Jaką wartość będzie miała zmienna `y`?

```
public class JakaWartosc {
  public static void main(String[] args) {
    int x = 5;
    int y = x + y;
  }
}
```

Ten program w ogóle się nie skompiluje. Do zmiennej `y` próbujemy przypisać wartość sumy zmiennych `x` oraz `y`. Zmienna `y` nie ma jeszcze przypisanej żadnej wartości, więc nie może być użyta do wyznaczenia swojej własnej wartości.

Kompilator wyświetli następujący komunikat:

```
JakaWartosc.java:4: error: variable y might not have been initialized
    int y = x + y;
                ^
1 error
```

---

14. Czy poniższy kod jest poprawny?

```
public class DwieZmienne {
  public static void main(String[] args) {
    int x = 5, byte y = 20;
  }
}
```

Nie – ten program w ogóle się nie skompiluje. Nie możemy zdefiniować na raz dwóch zmiennych o różnych typach – kompilator wyświetli następujący błąd:

```
DwieZmienne.java:3: error: <identifier> expected
    int x = 5, byte y = 20;
             ^
1 error
```

---

15. Co zostanie wypisane na ekran?

```
public class Test {
  public static void main(String[] args) {
    int a = 5 + b;
    int b = 10;

    System.out.println("a ma wartość: " + a);
  }
}
```

Ten kod w ogóle się nie skompiluje – definicja zmiennych jest niepoprawna. W linii:

```
int a = 5 + b;
```

próbujemy wyznaczyć wartość zmiennej `a` na podstawie wartości zmiennej `b`, jednakże w tym miejscu programu kompilator nie wie jeszcze czym jest „b”.

Kompilacja tego programu zakończy się następującym błędem:

```
Test.java:3: error: cannot find symbol
    int a = 5 + b;
                ^
  symbol:   variable b
  location: class Test
1 error
```

Poprawnie zapisany program powinien wyglądać jak poniżej:

```
public class Test {
  public static void main(String[] args) {
    int b = 10;
    int a = 5 + b;

    System.out.println("a ma wartość: " + a);
  }
}
```

---

16. Czy poniższy kod jest poprawny?

```
long duzaLiczba = 1000000000000;
```

Nie – liczby całkowite zapisane w kodzie źródłowym Java to literały, które są typu `int`. Wartość `1000000000000` przekracza zakres tego typu, więc nie może ona zostać zapisana w kodzie programu w takiej postaci. Kompilator zwróci następujący błąd:

```
LongDuzaLiczba.java:3: error: integer number too large
    long duzaLiczba = 1000000000000;
                      ^
1 error
```

Aby zapisać w kodzie programu wartość typu `long`, który ma większy zakres wartości niż typ `int`, należy na końcu liczby dodać małą bądź wielką literę `L`:

```
long duzaLiczba = 1000000000000L;
```

---

17. Czy poniższy kod jest poprawny?

```
float jednaDziesiata = 0.1;
```

Nie – literały zmiennoprzecinkowe (zawierające część ułamkową) zapisane w kodzie źródłowym programów języka Java są domyślnie typu `double`. Ten typ ma większy zakres wartości, niż typ `float`, więc kompilator zgłosi błąd kompilacji:

```
PrzypisanieDoFloat.java:3: error: incompatible types: possible lossy con-
version from double to float
    float jednaDziesiata = 0.1;
                           ^
1 error
```

Aby poprawnie przypisać do zmiennej typu `float` wartość z częścią ułamkową należy na jaj końcu dodać małą bądź wielką literę `f`:

```
float jednaDziesiata = 0.1f;
```

---

18. Czy poniższy kod jest poprawny?

```
byte z = 1_0_0;
```

Tak – w języku Java możemy stosować znak podkreślenia aby oddzielić od siebie kolejne części liczby, by była bardziej czytelna – porównaj: `1000000000` i `1_000_000_000`. W drugim przypadku użycie znaków podkreślenia pozwala na szybką identyfikację, że chodzi o jeden miliard.

---

19. Jaką wartość będzie miała zmienna `zagadka`?

```
int zagadka = -2147483648 - 1;
```

Wartością zmiennej `zagadka` będzie liczba `2147483647`. W tym przypadku dojdzie do tzw. „przekręcenia licznika” – liczba `-2147483648` to minimalna wartość, jaką mogą przechowywać zmienne typu `int`. Odjęcie jeden od tej liczby spowoduje, że zmiennej `zagadka` zostanie przypisana następna poprawna wartość z „drugiego końca” zakresu wartości typu `int`, którą w tym przypadku jest liczba `2147483647`.

20. Jaki jest typ zmiennej `jednaDruga`?

```
var jednaDruga = 0.5;
```

Gdy korzystamy ze słowa kluczowego `var`, typ zmiennej wyznaczany jest na podstawie wartości, która jest do niej przypisywana.

Typem zmiennej `jednaDruga` jest `double`, ponieważ typem literałów zmiennoprzecinkowych (tzn. liczb z częścią ułamkową) w kodzie źródłowym programów napisanych w Javie jest typ `double`.

4.2. Zadania

4.2.1. Dodawanie liczb

Napisz program, w którym zdefiniujesz trzy zmienne typu `int`. Do dwóch pierwszych przypisz dowolne liczby, a do trzeciej – wynik dodawania dwóch pierwszych liczb. Aby dodać do siebie wartości dwóch zmiennych, skorzystaj ze znaku `+` (plus). Wypisz na ekran komunikat z informacją o wykonanym sumowaniu i jaki jest jego wynik, np. „Suma liczb 5 i 8 to 13”.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_04/DodawanieLiczb.java

```
public class DodawanieLiczb {
  public static void main(String[] args) {
    int x, y, wynik; // 1

    x = 5;
    y = 10;
    wynik = x + y;

    System.out.println(
        "Suma liczb " + x + " i " + y + " to " + wynik
    );
  }
}
```

Na początku programu, w linii `(1)`, definiujemy trzy zmienne. Nadajemy im wartości w kolejnych liniach – zmiennym `x` oraz `y` przypisujemy konkretne liczby, a w zmiennej `wynik` zapiszemy wynik dodawania dwóch pozostałych zmiennych. Na końcu programu wyświetlamy komunikat dla użytkownika. Po uruchomieniu zobaczymy:

```
Suma liczb 5 i 10 to 15
```

4.2.2. Obwód trójkąta

Napisz program, który skorzysta z czterech zmiennych typu `float` w celu policzenia obwodu trójkąta. W trzech zmiennych zapisz długość każdego z boków, a do ostatniej zmiennej przypisz wynik – obwód trójkąta. Długości boków powinny być wartościami z częścią ułamkową (np. `10.5`, `7.9`, `12.1`). Wypisz na ekran informację

zawierającą długości boków oraz wyliczony obwód trójkąta. Wzór na obwód trójkąta to suma długości jego boków.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_04/ObwodTrojkata.java

```
public class ObwodTrojkata {
  public static void main(String[] args) {
    float a = 5.5f, b = 7.4f, c = 10.1f; // 1

    float obwodTrojkata = a + b + c;

    System.out.println(
        "Obwód trójkąta o bokach " + a + ", " + b + ", " +
        c + " wynosi " + obwodTrojkata
    );
  }
}
```

W linii `(1)` definiujemy trzy zmienne, które będą przechowywać długości boków trójkąta. Zgodnie z wymaganiami zadania, ich typem jest `float`. Aby przypisać w kodzie źródłowym Java liczbę z częścią ułamkową do zmiennej typu `float`, musimy na końcu tej liczby dopisać małą lub wielką literę `f` – w przeciwnym razie kod się nie skompiluje. Powodem jest to, że domyślnym typem liczb zmiennoprzecinkowych w Javie jest typ `double`, który ma większy zakres wartości, niż typ `float`. Dodanie na końcu liczby litery `f` powoduje, że dana wartość jest traktowana jako literał typu `float`, a nie `double`.

Czwartą zmienną definiowaną w tym programie jest `obwodTrojkata` – przypisujemy do niej wynik dodawania trzech pozostałych zmiennych, obliczając w ten sposób obwód trójkąta o długościach boków zapisanych w zmiennych `a`, `b`, `c`. Na końcu wypisujemy komunikat na ekran. Po uruchomieniu tego programu zobaczymy:

```
Obwód trójkąta o bokach 5.5, 7.4, 10.1 wynosi 23.0
```

4.2.3. Aktualna data

Napisz program, w którym do trzech różnych zmiennych przypiszesz aktualny dzień, miesiąc, i rok. Pamiętaj o odpowiednim nazewnictwie zmiennych. Wypisz na ekran wszystkie wartości, np. „Aktualny numer miesiąca to 4, numer dnia to 2, a rok to 2024”.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_04/AktualnaData.java

```
public class AktualnaData {
  public static void main(String[] args) {
    int dzien = 15;
    int miesiac = 4;
    int rok = 2024;

    System.out.println(
        "Aktualny numer miesiąca to " + miesiac +
        ", numer dnia to " + dzien + ", a rok to " + rok
    );
  }
}
```

Po uruchomieniu tego programu na ekranie zobaczymy komunikat:

```
Aktualny numer miesiąca to 4, numer dnia to 15, a rok to 2024
```

4.2.4. Liczba miesięcy w roku

Napisz program, w którym zdefiniujesz stałą, do której przypiszesz liczbę miesięcy w roku. Pamiętaj o odpowiednim nazewnictwie stałej. Wypisz wartość zdefiniowanej stałej na ekran, np. „Liczba miesięcy w roku to 12”.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_04/LiczbaMiesiecyWRoku.java

```
public class LiczbaMiesiecyWRoku {
  public static void main(String[] args) {
    final int LICZBA_MIESIECY_W_ROKU = 12;

    System.out.println(
       "Liczba miesięcy w roku to " + LICZBA_MIESIECY_W_ROKU
    );
  }
}
```

Aby zdefiniować stałą w języku Java, należy przed nazwą typu dodać słowo kluczowe `final`. Ponadto, nazwy stałych w Javie zapisujemy korzystając z konwencji SCREAMING_SNAKE_CASE, tzn. wszystkie litery w nazwie są wielkie, a poszczególne słowa są od siebie oddzielone znakiem podkreślenia. Po zdefiniowaniu stałej wypisujemy na ekran komunikat.

Po uruchomieniu tego programu zobaczymy:

```
Aktualny numer miesiąca to 4, numer dnia to 15, a rok to 2024
```

4.2.5. Inicjały

Napisz program, w którym przypiszesz swoje inicjały do dwóch zmiennych typu `char` – do każdej ze zmiennych po jednym znaku. Wypisz swoje inicjały na ekran – po każdej literze powinna następować kropka, np. „P. K.”.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_04/Inicjaly.java

```
public class Inicjaly {
  public static void main(String[] args) {
    char p = 'P';
    char k = 'K';

    System.out.println(p + "." + k + ".");
  }
}
```

Pojedyncze znaki, które przechowuje typ `char`, ujmujemy w apostrofy, w przeciwieństwie do stringów, czyli łańcuchów tekstowych, który zawierane są w cudzysłowie. Wypisując na ekran komunikat łączymy zmienne typu `char` wraz z łańcuchami tekstowymi zawierającymi kropkę. Wynik uruchomienia jest następujący:

```
P.K.
```

Czy możemy zamiast stringów z kropką użyć wartości typu `char`, to znaczy zapisać wyświetlanie danych na ekran tak, jak poniżej?

```
System.out.println(p + '.' + k + '.');
```

Tak, ale wyświetlony komunikat będzie zgoła inny:

```
247
```

Zmienne typu `char` reprezentują znaki przechowując ich wartości liczbowe. W tak zapisanym komunikacie do wyświetlenia nie zostanie użyty operator konkatenacji (łączenia) stringów, lecz operator dodawania – dodane zostaną liczbowe wartości każdego ze znaków, czyli wielkich liter `'P'` i `'K'` oraz, dwukrotnie, kropki `'.'`. Aby połączyć ze sobą znaki, należy użyć stringów, które zapisujemy w cudzysłowie. Dzięki temu operator `+` będzie operatorem konkatenacji stringów, a nie dodawania.

TYP STRING & UŻYWANIE TYPÓW ZŁOŻONYCH

5.1. Pytania

1. Do czego służy typ `String`?

Typ `String` służy do przechowywania *stringów*, czyli łańcuchów tekstowych.

---

2. Jakie są dwa sposoby na zapisanie w Javie literału typu `String`?

Podstawowy sposób na zapisanie łańcucha tekstowego to ujęcie tekstu w cudzysłów:

```
String komunikat = "Będzie padać - weź parasol.";
```

Od 15 wersji języka Java możemy także zapisywać w kodzie programów *bloki tekstu*, czyli łańcuchy tekstowe obejmujące kilka linii. Bloki tekstu ujmujemy w potrójny cudzysłów:

```
String blokTekstu = """
    Czwartek to "mały piątek".
    A środa to już bliżej niż dalej.""";
```

---

3. Jak połączyć ze sobą kilka wartości typu `String`?

Należy skorzystać z operatora konkatenacji stringów + jak w przykładzie poniżej:

```
// zmienna komunikat będzie zawierać string: Witaj Świecie!
String komunikat = "Witaj " + "Świecie!";
```

---

4. Jakie cechy mają wielolinijkowe łańcuchy tekstowe?

Bloki tekstu mają następujące cechy:

1. Po każdej linii tekstu dodawany jest znak nowej linii (poza ostatnią linią, o ile znaki cudzysłowu kończące blok tekstu są w tej samej linii),
2. Można sterować wcięciem na początku łańcucha tekstowego za pomocą odpowiedniego umiejscowienia znaków cudzysłowu, który kończą blok tekstu.
3. Znaki cudzysłowu nie muszą być „eskejpowane”.
4. Spacje z końców linii w blokach tekstu są usuwane. Aby wskazać, że na końcu linii ma się znajdować spacja, należy skorzystać ze znacznika `\s`.

Poniższy fragment kodu przedstawia opisane cechy bloków tekstu:

```
// 1
System.out.println("""
    Pierwsza linia.
    Druga linia."""
);
System.out.println("""
    Pierwsza linia.
    Druga linia.
    """ // kończące znaki """ są w osobnej linii, więc
);      // na końcu napisu 'Druga linia.' dodany zostanie
        // znak nowej linii, co widać poniżej
        // w przykładowym uruchomieniu tego kodu

// 2
System.out.println("""
    Pierwsza linia bez wcięcia.
    Druga linia bez wcięcia.
    """
);
System.out.println("""
    Pierwsza linia z wcięciem.
    Druga linia z wcięciem.
""" // końcowe znaki """ nie są wyrównane do początku
);  // tekstu, więc ten string będzie miał wcięcie

// 3
// nie trzeba „eskejpować” cudzysłowu za pomocą znaku \
System.out.println("""
    Znak cudzysłowu: "
    """);

// 4
System.out.print("""
    Policzę pole koła.
    Ile wynosi promień koła?\s
    """);

int promien = pobierzLiczbe();
```

Ten fragment kodu wypisuje na ekran następujące komunikaty:

```
Pierwsza linia.
Druga linia.
Pierwsza linia.
Druga linia.

Pierwsza linia bez wcięcia.
Druga linia bez wcięcia.

    Pierwsza linia z wcięciem.
    Druga linia z wcięciem.

Znak cudzysłowu: "

Policzę pole koła.
Ile wynosi promień koła? 7
```

Gdyby nie znak `\s` w ostatnim bloku tekstu, który jest wypisywany na ekran, to użytkownik wprowadzałby promień koła od razu po napisie:

```
Policzę pole koła.
Ile wynosi promień koła?7
```

5. Co wypisze na ekran poniższy fragment kodu?

```
String literaA = 'A';
System.out.println(literaA.toLowerCase());
```

Ten kod w ogóle się nie skompiluje – literały typu `String` w kodzie Java zapisujemy ujmując je w cudzysłów, a nie apostrofy. Kompilator zgłosi błąd:

```
PrzypiszA.java:3: error: incompatible types: char cannot be converted to
String
    String literaA = 'A';
                     ^
1 error
```

By kod był poprawny, do zmiennej `literaA` powinna zostać przypisana wartość `"A"`.

6. Co wypisze na ekran poniższy fragment kodu?

```
String str = "Czwartek to "mały piątek"";
System.out.println(str);
```

Ten kod w ogóle się nie skompiluje – aby umieścić w stringu znak cudzysłowu, należy go „wyeskejpować", tzn. poprzedzić go znakiem backslash `\`. Ten fragment kodu generuje następujące błędy podczas kompilacji:

```
WypiszKomunikat.java:3: error: ';' expected
    String str = "Czwartek to "mały piątek"";
                               ^
WypiszKomunikat.java:3: error: ';' expected
    String str = "Czwartek to "mały piątek"";
                                          ^
2 errors
```

Poprawnie zapisana inicjalizacja zmiennej `str` powinna wyglądać następująco:

```
String str = "Czwartek to \"mały piątek\"";
```

---

7. Ile pustych linii będzie dzielić komunikat `"Pierwsza linia"` i `"Druga linia"`?

```
System.out.print("Pierwsza linia.\n\n");
System.out.println("Druga linia.");
```

Jedna – instrukcja `print` nie dodaje automatycznie na końcu wypisywanego komunikatu znaku nowej linii, więc sumarycznie pierwsza linia zaprezentowanego kodu wypisze na ekran dwa znaki nowej linii (zapisane w komunikacie jako `\n\n`). Na ekranie zobaczymy:

```
Pierwsza linia. <- pierwszy znak \n jest na końcu tej linii
<- drugi znak \n jest jedynym znakiem (chociaż niewidocznym) w tej linii
Druga linia.
```

---

8. Co wypisze na ekran poniższy fragment kodu?

```
String str = """Pierwsza linia \
    Druga linia?""";
System.out.println(str);
```

Ten kod się nie skompiluje – treść bloku tekstu nie może zaczynać się w tej samej linii co otwierający go potrójny cudzysłów. Kompilator wyświetli komunikat:

```
WypiszBlokTekstu.java:3: error: illegal text block open delimiter se-
quence, missing line terminator
    String str = """Pierwsza linia \
                  ^
WypiszBlokTekstu.java:3: error: illegal character: '\'
    String str = """Pierwsza linia \
                                    ^
WypiszBlokTekstu.java:4: error: ';' expected
    Druga linia?""";
               ^
WypiszBlokTekstu.java:4: error: illegal text block open delimiter se-
quence, missing line terminator
    Druga linia?""";
                  ^
4 errors
```

Poprawnie zapisana inicjalizacja zmiennej `str` powinna wyglądać następująco:

```
String str = """
     Pierwsza linia \
     Druga linia?""";
```

---

9. W ilu liniach zostanie wypisany poniższy string?

```
String str = """
     Pierwsza linia.
     Druga linia. \
     Trzecia linia.
     """;
System.out.println(str);
```

W dwóch liniach:

```
Pierwsza linia.
Druga linia. Trzecia linia.
```

Dodanie znaku backslash `\` na końcu linii w bloku tekstu powoduje, że na końcu tej linii nie zostanie dodany automatycznie znak nowej linii.

10. Jak powinien być zapisany string z użyciem zwykłych stringów (tzn. zawartych w pojedynczym cudzysłowie) żeby odwzorować przedstawiony blok tekstu:

```
"""
    Pierwsza linia.
    Druga linia.
"""
```

Wypiszmy ten blok tekstu na ekran:

```
System.out.println(
    """
        Pierwsza linia.
        Druga linia.
    """
);
```

Efekt:

```
    Pierwsza linia.
    Druga linia.
```

Ponieważ potrójny cudzysłów zamykający ten blok tekstu jest w osobnej linii, to na końcu linii „Druga linia.” dodany zostanie znak nowej linii. Ponadto, ze względu na to, że ten zamykający potrójny cudzysłów nie jest wyrównany do początku tekstu, to ten tekst będzie miał wcięcie, co widać w komunikacie wypisanym na ekran.

Ten blok tekstu, zapisany za pomocą stringów ujętych w pojedynczy cudzysłów, powinien więc wyglądać następująco:

```
"    Pierwsza linia.\n" +
"    Druga linia.\n"
```

---

11. Do czego służą, i jakie argumenty przyjmują, metody typu `String`: `charAt`, `length`, `toLowerCase`, `toUpperCase`, `replace`, `substring`?

Opis metod:

- `charAt` – zwraca znak na danej pozycji w stringu; przyjmuje jeden argument – indeks znaku w stringu, który chcemy pobrać,

- `length` – zwraca liczbę znaków, z których string się składa; nie przyjmuje argumentów,
- `toLowerCase` i `toUpperCase` – służą do zwrócenia stringa, na rzecz którego zostały wywołane, z literami zamienionymi na małe litery (`toLowerCase`) lub na wielkie litery (`toUpperCase`); nie przyjmują argumentów,
- `replace` – zwraca kopię stringa, na rzecz którego jest wywoływana, ze wszystkimi wystąpieniami tekstu przesłanego jako pierwszy argument zastąpionymi tekstem podanym jako drugi argument,
- `substring` – zwraca fragment stringa, na rzecz którego jest wywoływana. Ten fragment zaczyna się od indeksu przesłanego jako pierwszy argument, a kończy na indeksie przesłanym jako drugi argument. Zwracany fragment nie zawiera znaku o indeksie wskazywanym jako drugi argument, tzn. zakres znaków wskazany przez dwa indeksy jest otwarty z prawej strony. Istnieje druga wersja tej metody, która przyjmuje tylko jeden argument – wtedy zwracany jest fragment od podanego indeksu aż do końca.

---

12. Który zapis jest poprawny?

```
String str = """
    Czwartek to "mały piątek".""";

String str2 = """
    Czwartek to \"mały piątek\".""";
```

Oba zapisy są poprawne, chociaż w drugim użycie znaków backslash `\` aby „wyeskejpować" znaki cudzysłowu jest nadmiarowe. W blokach tekstu możemy umieszczać znaki cudzysłowu bez potrzeby poprzedzania ich znakiem `\`.

---

13. Co zostanie wypisane na ekran w każdym z poniższych fragmentów kodu?

```
String str = "abc";
str.replace("a", "A");
System.out.println(str);
```

Wypisany zostanie ciąg znaków:

```
abc
```

Metoda `replace` nie zmienia stringa, na rzecz którego jest wywoływana, lecz zwraca zmienioną kopię. W tym przypadku tej zwróconej wartości – stringa `"Abc"` – do niczego nie przypisujemy. Wartość zmiennej `str` pozostaje niezmieniona.

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

Wypisane zostanie 0, ponieważ pusty string nie zawiera żadnych znaków.

```
System.out.println("Witaj!".charAt("Witaj!".length()));
```

Wykonanie tego fragment kodu spowoduje zakończenie się programu błędem:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 6 out
of bounds for length 6
  at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:55)
  at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:52)
  at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:213)
  at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:210)
  at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:98)
  at     java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Precondi-
tions.java:106)
  at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
  at java.base/java.lang.String.checkIndex(String.java:4881)
  at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:46)
  at java.base/java.lang.String.charAt(String.java:1582)
  at CharAtILength.main(CharAtILength.java:3)
```

Indeks ostatniego znaku jest o jeden mniejszy niż długość stringa, ponieważ numeracja znaków w łańcuchach tekstowych zaczyna się od `0`, a nie `1`. Podając metodzie `charAt` indeks `6` (długość stringa) wyszliśmy poza jego zakres – indeks ostatniego znaku to `5`.

```
String str = "Ala ma kota";
System.out.println(str.replace(" ", ""));
```

Ten fragment kodu wyświetli na ekranie:

```
Alamakota
```

Zapisanie argumentów metody `replace` w ten sposób powoduje „wycięcie” ze stringa znaków spacji.

```
String str = "Ala ma kota";
System.out.println(str.replace("ala", "Jola"));
```

Ten fragment kodu wyświetli na ekranie:

```
Ala ma kota
```

Małe i wielkie litery są traktowane jako różne znaki. Metoda `replace` nie znalazła w treści zmiennej `str` fragmentu `"ala"`.

```
String tekst = "Ala ma kota";
System.out.println(tekst.substring(tekst.length()));
```

Ten fragment kodu wypisze na ekran pusty komunikat – metoda `substring` zwróci w tym przypadku fragment stringa zaczynający się od końca stringa – skoro to koniec, to nie ma tam żadnych znaków.

```
String str = "programowanie";
String fragment = str.substring(3, 6).replace("m", "cz");
System.out.println(fragment);
```

Ten fragment kodu wyświetli na ekran komunikat:

```
gra
```

Metoda `replace` nie ma w tym przypadku efektu – metoda `substring(3, 6)` zwróci fragment `"gra"`, bez litery `m`, ponieważ drugi argument przekazywany do metody `substring` wyznacza zakres otwarty z prawej strony.

```
System.out.println("Witaj".length);
```

Ten fragment kodu się nie skompiluje – po nazwie metody `length` brakuje nawiasów, które są wymagane, aby wywołać metodę. Kompilator wyświetli błąd:

```
LengthBezNawiasow.java:3: error: cannot find symbol
    System.out.println("Witaj".length);
                               ^
  symbol:   variable length
  location: class String
1 error
```

---

14. Jaki jest indeks pierwszego znaku w wartościach typu `String`?

W języku Java pierwszym indeksem znaków w stringach jest `0`.

---

15. Jak nazywa się klasa złożona odwzorowująca typ prymitywny `int`?

Ta klasa to `Integer`.

---

16. Czym różni się użycie metody `String.format`, metody `formatted`, oraz `System.out.printf`?

Te metody mają następujące cechy:

- `String.format` – przyjmuje komunikat wraz z wartościami, które mają zastąpić specjalne znaczniki umieszczone w komunikacie, i zwraca łańcuch tekstowy zawierający te wartości,

- `"pewien string".formatted` – ta metoda wywoływana jest bezpośrednio na stringu lub zmiennej typu `String` – przyjmuje argumenty, które mają zastąpić znaczniki zawarte w stringu, i zwraca kopię tego stringa z umieszczonymi w nim wartościami; dostępna od 15 wersji języka Java,
- `System.out.printf` – wypisuje na ekran komunikat, w którym specjalne znaczniki będą zastąpione argumentami przesłanymi do tej metody.

---

17. Do czego w metodach formatujących tekst służą znaczniki `%d`, `%s`, `%f` ?

Te znaczniki określają miejsca w stringu, w których ma być wstawiona pewna wartość. Za ich pomocą można także zmieniać sposób wyświetlania danych, np. dla liczb z częścią ułamkową możemy określić ile miejsc po przecinku ma zostać użytych. Znacznik `%d` jest zastępowany podaną liczbą całkowitą, `%s` – wartością tekstową, a `%f` liczbą zmiennoprzecinkową.

---

18. Jak przepisać poniższy fragment kodu aby korzystał z `System.out.println` i by na ekran został wyświetlony taki sam komunikat?

```
System.out.print("Witaj!\n\n");
```

Taki sam komunikat wypiszemy za pomocą metody `println` w następujący sposób:

```
System.out.println("Witaj!\n");
```

Drugi znak nowej linii zostanie automatycznie wypisany przez metodę `println`.

---

19. Przepisz trzy razy poniższy kod, aby korzystał z metody `String.format`, `formatted`, oraz `System.out.printf`, i wypisywał na ekran ten sam komunikat:

```
String dzienTygodnia = "poniedziałek";
int godzina = 12;
int minuta = 55;

System.out.println(
    "Witaj!" + "\nDzisiaj jest " +
    dzienTygodnia + ".\nAktualna godzina to " +
    godzina + ":" + minuta + "."
);
```

Ten fragment kodu wypisuje na ekran następujący komunikat:

```
Witaj!
Dzisiaj jest poniedziałek.
Aktualna godzina to 12:55.
```

Zapis używający metody `String.format`:

```
System.out.println(
  String.format(
    "Witaj!\nDzisiaj jest %s.\nAktualna godzina to %d:%d.",
    dzienTygodnia, godzina, minuta
  )
);
```

Wykorzystanie metody `formatted`:

```
String komunikat =
    "Witaj!\nDzisiaj jest %s.\nAktualna godzina to %d:%d.";

System.out.println(
    komunikat.formatted(dzienTygodnia, godzina, minuta)
);
```

Metoda `formatted` może też zostać wywołana bezpośrednio na literale typu `String`.

Użycie `System.out.printf`:

```
System.out.printf(
  "Witaj!\nDzisiaj jest %s.\nAktualna godzina to %d:%d.\n",
  dzienTygodnia, godzina, minuta
);
```

W tym przypadku należało jeszcze dodać na koniec znak nowej linii `\n`, ponieważ metoda `printf` nie dodaje go automatycznie, w przeciwieństwie do metody `println`.

---

20. Czym różnią się od siebie metody `System.out.print` i `System.out.println`?

Metoda `println` dodaje na koniec wypisywanego komunikatu znak nowej linii. Dzięki temu kolejne linie tekstu będą zaczynać się od nowej linii na ekranie. Metoda `print` nie dodajemy znaku nowej linii – przydaje się ona, gdy chcemy wypisać kilka komunikatów jeden obok drugiego.

21. Jeżeli chcemy wczytać od użytkownika dane za pomocą klasy `Scanner`, to co musimy dodać na początek programu?

Na początek programu należy dodać instrukcję `import`, która zaimportuje do programu klasę `Scanner`:

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

5.2. Zadania

5.2.1. Pobrane słowa w odwrotnej kolejności

Napisz program, który wczyta od użytkownika trzy słowa i wypisze je w odwrotnej kolejności, niż podał je użytkownik, oddzielone przecinkami. Dla przykładu, gdy użytkownik poda słowa `Ala ma kota`, to program wyświetli `kota, ma, Ala`.

Do pobrania od użytkownika słów użyjemy metody `pobierzSlowo`:

```
public static String pobierzSlowo() {
  return new Scanner(System.in).next();
}
```

Użycie klasy `Scanner` wymaga dodania na początku programu instrukcji `import`:

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

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_05/PobraneSlowaWOdwrotnejKolejnosci.java

```
import java.util.Scanner;

public class PobraneSlowaWOdwrotnejKolejnosci {
  public static void main(String[] args) {
    System.out.print("Podaj pierwsze słowo: ");
    String slowo1 = pobierzSlowo();

    System.out.print("Podaj drugie słowo: ");
    String slowo2 = pobierzSlowo();

    System.out.print("Podaj trzecie słowo: ");
    String slowo3 = pobierzSlowo();

    System.out.printf(
        "%s, %s, %s", slowo3, slowo2, slowo1
    );
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

W metodzie `main` najpierw prosimy użytkownika o podanie trzech słów, a następnie wypisujemy je w odwrotnej kolejności, formatując komunikat używając metody `printf` oraz znaczników `%s`, które zostaną zastąpione wartościami zmiennych przekazanych do metody `printf`. Przykładowe uruchomienie tego programu:

```
Podaj pierwsze słowo: Ala
Podaj drugie słowo: ma
Podaj trzecie słowo: kota
kota, ma, Ala
```

5.2.2. Pole i obwód kwadratu o podanym boku

Napisz program, w którym pobierzesz od użytkownika liczbę, i wypiszesz na ekran pole i obwód kwadratu, który posiada bok o podanej długości.

Wzór na pole kwadratu to `długość boku * długość boku`, natomiast wzór na obwód to `4 * długość boku`. Wypisz na ekran ładnie sformatowany komunikat z wyliczonym polem i obwodem kwadratu.

W tym programie użyjemy metody `pobierzLiczbe` do pobrania od użytkownika długości boku kwadratu:

```
public static int pobierzLiczbe() {
  return new Scanner(System.in).nextInt();
}
```

Aby skorzystać z klasy `Scanner`, musimy ją zaimportować:

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

Proponowane rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_05/PoleIObwodKwadratu.java

```
import java.util.Scanner;

public class PoleIObwodKwadratu {
  public static void main(String[] args) {
    System.out.print("Podaj długość boku kwadratu: ");
    int bok = pobierzLiczbe();

    int poleKwadratu = bok * bok;
    int obwodKwadratu = 4 * bok;
```

```
    System.out.printf(
      "Pole kwadratu o boku %d to %d, a jego obwód to %d.",
      bok, poleKwadratu, obwodKwadratu
    );
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` prosimy użytkownika o podanie długości boku kwadratu. Następnie, używamy wczytanej wartości do obliczenia pola i obwodu koła, które zapisujemy w zmiennych `poleKwadratu` oraz `obwodKwadratu`. Na końcu metody `main` wyświetlamy komunikat dla użytkownika:

```
Podaj długość boku kwadratu: 10
Pole kwadratu o boku 10 to 100, a jego obwód to 40.
```

5.2.3. Czy słowo kończy się na literę?

Napisz program, w którym pobierzesz od użytkownika tekst i wypiszesz na ekran informację `true` jeżeli ostatni znak w tym tekście to litera, a w przeciwnym razie `false`. Do wykonania tego sprawdzenia możesz użyć metody `Character.isLetter`. Zwraca ona wartość `true` jeżeli przekazany do niej znak jest literą, a w przeciwnym razie `false`.

Do pobrania tekstu użyjemy metody `pobierzSlowo`, z której korzystaliśmy już w rozwiązaniu zadania „5.2.1 Pobrane słowa w odwrotnej kolejności”:

Pytania_i_zadania_rozwiazania/Rozdzial_05/CzySlowoKonczySieNaLitere.java

```
import java.util.Scanner; // 1

public class CzySlowoKonczySieNaLitere {
  public static void main(String[] args) {
    System.out.print("Podaj słowo: ");
    String slowo = pobierzSlowo();

    int indeksOstatniegoZnaku = slowo.length() - 1; // 2

    System.out.println(
        Character.isLetter( // 3
            slowo.charAt(indeksOstatniegoZnaku) // 4
```

```
            )
        );
    }

    public static String pobierzSlowo() {
        return new Scanner(System.in).next();
    }
}
```

Na początku metody `main` pobieramy od użytkownika słowo za pomocą metody `pobierzSlowo`. Ze względu na użycie w niej klasy `Scanner` należy do programu dodać stosowną instrukcję `import` (linia `(1)`).

W linii `(2)` zdefiniowałem pomocniczą zmienną `indeksOstatniegoZnaku`. Posłuży ona jako argument dla metody `charAt` w linii `(4)`, aby pobrać ostatni znak w odczytanym słowie. Indeks ostatniego znaku w stringu to `długość_stringa - 1`, ponieważ numeracja znaków zaczyna się od `0`, a nie `1`.

Metoda `charAt` zwróci ostatni znak, który zostanie przekazany do metody `Character.isLetter` (linia `(3)`). Ta metoda zwraca wartość `true` gdy przekazany jej znak jest literą, a w przeciwnym razie `false`.

Dwa przykładowe uruchomienia tego programu:

```
Podaj słowo: kot
true
```

```
Podaj słowo: Pada!
false
```

5.2.4. Użycie indexOf

Napisz program, w którym pobierzesz od użytkownika dwa słowa. Następnie, jeżeli drugie słowo zawiera się w pierwszym, wypisz indeks, od którego to drugie słowo zaczyna się w pierwszym słowie. W tym celu skorzystaj z metody `indexOf` – zajrzyj do dokumentacji aby dowiedzieć się, jak działa ta metoda. Jeżeli drugie słowo nie jest zawarte w pierwszym, to program powinien wypisać na ekran liczbę `-1` (taką właśnie wartość zwraca metoda `indexOf`, gdy dany string nie zostanie znaleziony).

Dla przykładu – jeżeli użytkownik poda słowo `programowanie` i `gram`, to program powinien wypisać na ekran liczbę `3`, bo słowo `gram` jest zawarte w słowie `programowanie`, i zaczyna się pod trzecim indeksem.

Jeżeli użytkownik poda słowa `programowanie` i `nauka`, to program powinien wyświetlić `-1`, bo słowo `nauka` nie jest zawarte w słowie `programowanie`.

Metoda `indexOf` typu `String` przyjmuje argument – łańcuch tekstowy, który chcemy znaleźć w stringu, na rzecz którego ta metoda jest wywoływana. Jeżeli zostanie on znaleziony, to zwrócony zostanie indeks, gdzie ten fragment się zaczyna. W przeciwnym razie metoda `indexOf` zwróci liczbę `-1`. Opis tej metody znajdziesz w oficjalnej dokumentacji Biblioteki Standardowej Java:

https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/String.html#indexOf(java.lang.String)

Spójrzmy na proponowane rozwiązanie korzystające z tej metody:

Pytania_i_zadania_rozwiazania/Rozdzial_05/UzycieIndexOf.java

```
import java.util.Scanner;

public class UzycieIndexOf {
  public static void main(String[] args) {
    System.out.print("Podaj pierwsze słowo: ");
    String slowo1 = pobierzSlowo();

    System.out.print("Podaj drugie słowo: ");
    String slowo2 = pobierzSlowo();

    System.out.println(slowo1.indexOf(slowo2)); // 1
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

W pierwszej kolejności pobieramy od użytkownika dwa słowa, po czym w linii `(1)` wywołujemy na pierwszym z nich metodę `indexOf`, do której jako argument przekazujemy drugie słowo. Jeżeli drugie słowo zostanie znalezione w pierwszym, to metoda `indexOf` zwróci indeks, gdzie to drugie słowo zaczyna się w pierwszym:

```
Podaj pierwsze słowo: programowanie
Podaj drugie słowo: gram
3
```

Jeżeli słowo nie zostanie znalezione, to zwrócona zostanie liczba `-1`:

```
Podaj pierwsze słowo: programowanie
Podaj drugie słowo: nauka
-1
```

5.2.5. Liczenie pierwiastka kwadratowego

Napisz program, który pobierze od użytkownika liczbę, policzy z niej pierwiastek kwadratowy, i wyświetli na ekran ładnie sformatowany wynik. Pierwiastek kwadratowy możesz policzyć za pomocą metody `Math.sqrt`, która przyjmuje jeden argument. Wyświetlony wynik powinien być zaokrąglony do dwóch miejsc po przecinku (skorzystaj z odpowiednio zapisanego znacznika `%f` w metodzie takiej jak `System.out.printf` lub `String.format`).

Klasa `Math` z Biblioteki Standardowej Java zawiera przydatne metody służące do wykonywania różnego rodzaju obliczeń, takich jak wyznaczenie pierwiastka kwadratowego pewnej liczby. Ta klasa wchodzi w skład pakietu `java.lang`, który jest automatycznie importowany do programów napisanych w języku Java, więc nie musimy na początku programu dodawać instrukcji `import`.

Jednakże, korzystając z metody `pobierzLiczbe`, która używa klasy `Scanner`, musimy dodać na początek instrukcję `import`, która zaimportuje klasę `Scanner`:

Pytania_i_zadania_rozwiazania/Rozdzial_05/PierwiastekKwadratowy.java

```
import java.util.Scanner;

public class PierwiastekKwadratowy {
  public static void main(String[] args) {
    System.out.print("Podaj liczbę: ");
    int liczba = pobierzLiczbe();

    double pierwiastekLiczby = Math.sqrt(liczba); // 1

    System.out.printf(
        "Pierwiastek kwadratowy liczby %d wynosi %.2f.",
        liczba,
        pierwiastekLiczby
    );
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` wczytujemy od użytkownika liczbę. W linii `(1)` wyznaczamy jej pierwiastek kwadratowy za pomocą metody `Math.sqrt`, a wynik przypisujemy do zmiennej `pierwiastekLiczby`.

Wartość tej zmiennej wypiszemy w komunikacie dla użytkownika korzystając ze znacznika `%.2f`, dzięki któremu liczba będzie wyświetlona z dwoma miejscami po przecinku. Przykładowe uruchomienie tego programu:

```
Podaj liczbę: 99
Pierwiastek kwadratowy liczby 99 wynosi 9,95.
```

5.2.6. Nazwa w konwencji SCREAMING_SNAKE_CASE

Napisz program, który pobierze od użytkownika trzy słowa i zapisze je w konwencji SCREAMING_SNAKE_CASE, tzn. wszystkie litery w każdym słowie powinny zostać zamienione na wielkie litery, a słowa powinny zostać od siebie oddzielone znakiem podkreślenia. Wypisz na ekran ładnie sformatowaną informację, w której zawrzesz podane przez użytkownika słowa oraz powstałą z nich nazwę zapisaną w konwencji SCREAMING_SNAKE_CASE. Wypisywana nazwa powinna być ujęta w cudzysłów.

Przykład: jeżeli użytkownika poda słowa `moja`, `PrzykladowA`, `nAZWa`, to program powinien z nich stworzyć następującą nazwę: `MOJA_PRZYKLADOWA_NAZWA`

Program rozwiązujący to zadanie może wyglądać następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_05/NazwaWScreamingSnakeCase.java

```
import java.util.Scanner;

public class NazwaWScreamingSnakeCase {
  public static void main(String[] args) {
    System.out.print("Podaj pierwsze słowo: ");
    String slowo1 = pobierzSlowo();

    System.out.print("Podaj drugie słowo: ");
    String slowo2 = pobierzSlowo();

    System.out.print("Podaj trzecie słowo: ");
    String slowo3 = pobierzSlowo();

    String nazwaWScreamingSnakeCase =
        slowo1.toUpperCase() + "_" +
        slowo2.toUpperCase() + "_" +
        slowo3.toUpperCase(); // 1

    System.out.printf("""
        Nazwa w konwencji SCREAMING_SNAKE_CASE \
```

```
            składające się ze słów:
            \t%s
            \t%s
            \t%s
            to: "%s"
            """,
            slowo1, slowo2, slowo3, nazwaWScreamingSnakeCase
        );
    }

    public static String pobierzSlowo() {
        return new Scanner(System.in).next();
    }
}
```

Na początku metody `main` wczytujemy od użytkownika słowa i zapisujemy je w trzech zmiennych. W linii `(1)` budujemy z nich nazwę zapisaną w konwencji `SCREAMING_SNAKE_CASE`. Aby zamienić wszystkie litery w słowach na wielkie, korzystamy z metody `toUpperCase`. Na końcu metody `main` przygotowujemy ładnie sformatowany komunikat do wypisania dla użytkownika.

Przykładowy wynik uruchomienia tego programu jest następujący:

```
Podaj pierwsze słowo: moja
Podaj drugie słowo: PrzykladowA
Podaj trzecie słowo: nAZWa
Nazwa w konwencji SCREAMING_SNAKE_CASE składające się ze słów:
      moja
      PrzykladowA
      nAZWa
to: "MOJA_PRZYKLADOWA_NAZWA"
```

OPERATORY & RZUTOWANIE

6.1. Pytania

1. Czy wszystkie operatory mają takie same priorytety?

Nie – operatory mają różne priorytety. Dla przykładu, operator mnożenia `*` ma wyższy priorytet, niż operator dodawania `+`, więc w wyrażeniu `2 + 2 * 2` w pierwszej kolejności wykonane zostanie mnożenie.

---

2. Czy można zmieniać priorytet operatorów?

Można, korzystając z nawiasów. Aby w wyrażeniu `2 + 2 * 2` najpierw wykonać dodawanie, należy zapisać je z nawiasami w następujący sposób: `(2 + 2) * 2`.

---

3. Jeżeli operatory mają taki sam priorytet, to co wyznacza kolejność operacji, które będą wykonywane?

W takim przypadku operacje zostaną wykonane w kolejności:

- od lewej do prawej, jeżeli operatory są lewostronnie łączne,
- od prawej od lewej, gdy operatory są prawostronnie łączne.

---

4. Jaki wynik daje użycie operatora dzielenia `/` gdy jego argumenty (operandy) są liczbami całkowitymi?

Operatora dzielenia `/` daje w wyniku liczbę całkowitą, zaokrągloną w dół, gdy oba jego argumenty są liczbami całkowitymi, np. wartość wyrażenia `10 / 4` wynosi `2`.

---

5. Co to jest rzutowanie typów?

Rzutowanie typów to traktowanie wartości jednego typu jako wartości innego typu. Aby wykonywać rzutowanie, zapisujemy oczekiwany typ w nawiasach przed wartością, np. `(double) zmiennaTypuInt` spowoduje, że wartość zmiennej `zmiennaTypuInt` będzie traktowana jako liczba typu `double`.

6. Jak otrzymać w wyniku dzielenia liczbę zmiennoprzecinkową?

Mamy dwie opcje:

- możemy w wyrażeniu użyć zmiennej lub literału typu `double` bądź `float`,
- możemy skorzystać z rzutowania.

W drugiej linii poniżej pierwszy argument operatora dzielenia `/` to literał typu `double`, więc wynikiem dzielenia będzie wartość typu `double`. W trzeciej linii korzystamy z rzutowania:

```
// wypisze 2 - dzielenie calkowite
System.out.println(10 / 4);
System.out.println(10.0 / 4); // wypisze 2.5
System.out.println((double) 10 / 4); // wypisze 2.5
```

7. Do czego służą operatory `+=`, `-=`, `*=` itp.?

Są to skrótowe operatory przypisania, np. operator `+=` powoduje dodanie do zmiennej po lewej stronie operatora wartości wyrażenia po jego prawej stronie:

```
int a = 5;
a += 10; // zmienna a będzie miała wartość 5 + 10, czyli 15
```

8. Do czego służą operatory `++` i `--` i czym się różnią ich pre- i postfixowe wersje?

Te operatory służą do zwiększania oraz zmniejszania wartości zmiennej o `1`. Postfixowe wersje tych operatorów różnią się tym od prefixowych, że zwracają oryginalną wartość zmiennej (wartość przed zmianą). Prefixowe wersje najpierw zmieniają wartość zmiennej, a potem tę (nową) wartość zwracają:

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

// wypisze 2, bo operator postfixowy ++ zwiększy wartość
// zmiennej z 2 do 3, a potem zwróci oryginalną wartość
// tej zmiennej (tzn. przed zwiększeniem o jeden)
System.out.println(x++);
```

```
// wypisze 3, bo operator prefixowy ++ najpierw zwiększy
// wartość zmiennej z 2 do 3, a potem zwróci tę wartość
System.out.println(++y);
```

---

9. Ile argumentów przyjmuje operator plus `+` ?

Znak plus `+` jest używany do zapisu zarówno dwuargumentowych operatorów dodawania i konkatenacji stringów, jak i do zapisu jednoargumentowego operatora wskazującego, że dana wartość jest dodatnia:

```
int suma = 5 + 10;
String komunikat = "Witaj" + " Świecie!";
int x = +5;
```

---

10. Jaką wartość będzie miała zmienna `y`?

```
int x = 0;
int y = x++ * (5 + 2);
```

Zmiennej `y` zostanie nadana wartość `0`. Wartość wyrażenia `x++` to oryginalna wartość zmiennej `x`, ponieważ użyty został operator postfixowy. Oznacza to, że wynik wyrażenia `(5 + 2)` zostanie pomnożony przez `0` (czyli przez wartość zmiennej `x` przed zmianą wywołaną przez operator `++`).

---

11. Jaką wartość będzie miała zmienna `x`?

```
int x = (int) "1";
```

Ten fragment kodu powoduje błąd kompilacji: „incompatible types: String cannot be converted to int”. Nie można zrzutować wartości typu `String` na wartość typu `int`.

---

12. Jaką wartość będzie miała zmienna `b`?

```
byte b = 10;
b += 23.7;
```

Wartością zmiennej `b` będzie liczba `33`. Pomocnicze operatory przypisania wykonują automatyczne rzutowanie wartości wyrażenia, które zostanie obliczone w ramach działania danego pomocniczego operatora przypisania, na typ zmiennej, która jest zmieniana. W tym przypadku wynik wyrażenia `10 + 23.7` zostanie zrzutowany na wartość typu `byte`, więc kompilator nie zgłosi błędu związanego z niemożliwością dodania wartości zmiennoprzecinkowej do zmiennej typu całkowitego.

Korzystając ze zwykłego operatora przypisania ten fragment kodu można zapisać w następujący sposób:

```
byte b = 10;
b = (byte) (b + 23.7);
```

---

13. Jaki będzie wynik działania `10 % 10` ?

Wynikiem będzie `0`, ponieważ dzielenie liczby przez samą siebie nie pozostawia żadnej reszty z dzielenia.

---

14. Jaką wartość będzie miała zmienna `i`?

```
char c = '1';
int i = c + 1;
```

Zmienna `i` będzie miała wartość `50`, ponieważ wartość zmiennej `c` zostanie zamieniona na liczbowy kod znaku `'1'`, który wynosi `49`. Ta liczba zostanie dodana do `1`, więc do zmiennej `i` przypisana zostanie liczba `50`.

---

15. Jaką wartość będzie miała każda ze zmiennych?

```
double x = 5 / 2;
double y = 5 / (int) 2.1;
int z = 9 / 4.5;
```

Ten fragment kodu spowoduje błąd kompilacji, ponieważ do zmiennej `z` próbujemy przypisać wynik dzielenia zmiennoprzecinkowego – wynikiem wyrażenia `9 / 4.5` jest wartość typu `double`, która nie może zostać przypisana do zmiennej typu `int`.

Gdyby pominąć definicję zmiennej `z`, to wartością zmiennej `x` byłaby liczba `2`, ponieważ w przypisywanym wyrażeniu użyte zostały liczby całkowite, więc wykonane zostanie dzielenie całkowite. Podobnie w przypadku zmiennej `y`, której wartością także będzie `2`. Chociaż jedna z wartości użytych w wyrażeniu jest typu `double`, to jest ona najpierw rzutowana na wartość typu `int`, więc ponownie wykonane zostanie dzielenie całkowite (pomijające część ułamkową wyniku).

---

16. Jakie komunikaty zostaną wypisane na ekran?

```
System.out.println("10 + 3 to: " + 10 + 3);
System.out.println("10 * 3 to: " + 10 * 3);
System.out.println("10 - 3 to: " + 10 - 3);
```

Pierwsza linia wyświetli na ekran komunikat `10 + 3 to: 103`, ponieważ operator konkatenacji stringów i operator dodawania mają taki sam priorytet – związane z nimi operacje będą wykonywane od lewej do prawej, ponieważ te operatory są lewostronnie łączne. Najpierw do stringa `"10 + 3 to: "` dodana zostanie liczba `10`, dając w wyniku string `"10 + 3 to: 10"`, a następnie doklejona zostanie jeszcze liczba `3`, dając finalnie wynikowy string `"10 + 3 to: 103"`.

W przypadku drugiej instrukcji `println`, wypisany zostanie tekst `10 * 3 to: 30`. Operator mnożenia ma wyższy priorytet niż operator konkatenacji stringów, więc najpierw obliczona zostanie wartość wyrażenia `10 * 3`, a potem ten wynik zostanie doklejony do stringa `"10 * 3 to: "`.

Ostatnia linia spowoduje błąd kompilacji „bad operand types for binary operator '-'". Podobnie, jak w przypadku pierwszej linii, operacje także będą wykonywane od lewej do prawej, bo operator odejmowania ma taki sam priorytet, jak operator konkatenacji stringów. Najpierw do stringa `"10 - 3 to: "` zostałaby dodana liczba `10`, dając w wyniku string `"10 - 3 to: 10"`. Pozostałaby jeszcze do wykonania operacja `"10 - 3 to: 10" - 3`, która nie ma sensu – kompilator wychwyci ten problem i zgłosi błąd kompilacji.

17. Jaka będzie wartość zmiennej `x`?

```
int y = 10;
int x += y;
```

Ten fragment kodu spowoduje błąd kompilacji – pomocniczy operator przypisania nie może być stosowany podczas inicjalizacji zmiennej – zmienna `x` nie ma jeszcze żadnej wartości w tym przypadku.

---

18. Jaka będzie wartość zmiennej `x`?

```
int x = (x = 10) * 5;
```

Wartością zmiennej `x` będzie liczba `50`. Najpierw wykonana zostanie operacja z nawiasów, która zainicjalizuje zmienną `x` liczbą `10`. Następnie, ta wartość zostanie pomnożona przez `5`, więc finalnie do zmiennej `x` zostanie przypisana liczba `50`.

---

19. Jak uzyskać ten sam wynik korzystając ze zwykłego operatora przypisania zamiast operatora `+=`?

```
int x = 10;
double y = 3.5;
int z = 1;

z += x * y;
```

Przypisanie wartości do zmiennej `z` korzystające ze zwykłego operatora przypisania wygląda następująco:

```
z = (int) (z + (x * y));
```

---

20. Jaka będzie wartość zmiennej `liczba`?

```
int liczba = (int) 99.9999;
```

Zmiennej `liczba` zostanie przypisana wartość `99`. Rzutowanie wartości zmiennoprzecinkowej na liczbę całkowitą powoduje pominięcie części ułamkowej, a zaokrąglenia nie są wykonywane.

---

21. Jaką wartość będzie miała każda ze zmiennych?

```
int x = 5++;
int y = ++5;
```

Ten kod się nie skompiluje, ponieważ operatory inkrementacji (a także dekrementacji) oczekują zmiennych jako swoich argumentów, a nie literałów.

---

22. Jaką wartość będzie miała zmienna `liczba`?

```
double liczba = 15 (double) / 10;
```

Ten kod się nie skompiluje, ponieważ rzutowanie zostało wykonane w nieprawidłowy sposób. Aby zrzutować wartość na pewien typ, ten typ musi być umieszczony przed wartością, a nie za nią, jak w tym przypadku.

---

23. Jak poprawić poniższy fragment kodu, aby wyświetlał komunikat „10 - 9 to 1”?

```
System.out.println("10 - 9 to " + 10 - 9);
```

Odejmowanie należy ująć w nawiasy, aby zostało wykonane jako pierwsze:

```
System.out.println("10 - 9 to " + (10 - 9));
```

6.2. Zadania

6.2.1. Liczba o przeciwnym znaku

Napisz program, w którym pobierzesz od użytkownika liczbę i wypiszesz ją z przeciwnym znakiem. Dla przykładu, jeżeli użytkownik poda liczbę `5`, to program wypisze na ekran liczbę `-5`, a dla `-5` program wypisze `5`.

W tym programie należy skorzystać z jednoargumentowego operatora minus `-`, który zmienia znak danej liczby na przeciwny:

Pytania_i_zadania_rozwiazania/Rozdzial_06/LiczbaOPrzeciwnymZnaku.java

```
import java.util.Scanner;

public class LiczbaOPrzeciwnymZnaku {
  public static void main(String[] args) {
    System.out.print("Podaj liczbę: ");
    int liczba = pobierzLiczbe();

    System.out.println(-liczba); // 1
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` pobieramy od użytkownika liczbę. Korzystamy w tym celu z metody `pobierzLiczbe`, która używa klasy `Scanner`, więc na początku programu znajduje się instrukcja `import`, która tę klasę importuje.

W linii `(1)` wypisujemy z powrotem pobraną liczbę na ekran, ale zmieniamy jej znak za pomocą jednoargumentowego operatora minus `-`.

Dwa przykładowe uruchomienia tego programu:

```
Podaj liczbę: -5
5
```

```
Podaj liczbę: 5
-5
```

6.2.2. Liczba dni do końca roku

Napisz program, w którym pobierzesz od użytkownika numer miesiąca (od `1` do `12`) i numer dnia miesiąca (od `1` do `30`). Policz ile dni zostało jeszcze do końca roku. Ze względu na to, że nie omówiliśmy jeszcze instrukcji warunkowych, przyjmij następujące założenia:

- miesiące mają po 30 dni,
- rok ma 360 dni.

Wypisz na ekran ile dni zostało do końca roku.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_06/LiczbaDniDoKoncaRoku.java

```
import java.util.Scanner;

public class LiczbaDniDoKoncaRoku {
  public static void main(String[] args) {
    System.out.print("Podaj numer miesiąca: ");
    int miesiac = pobierzLiczbe();

    System.out.print("Podaj numer dnia miesiąca: ");
    int dzien = pobierzLiczbe();

    int ileDniMinelo = (miesiac - 1) * 30 + dzien; // 1
    int liczbaDniaDoKoncaRoku = 360 - ileDniMinelo; // 2

    System.out.println(
      "Liczba dnia do końca roku: " + liczbaDniaDoKoncaRoku
    );
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` pobieramy aktualny numer miesiąca i dnia miesiąca. Następnie, w linii `(1)`, obliczamy ile dni już minęło w tym roku. Zastosowany przeze mnie wzór najpierw liczy ile pełnych miesięcy mamy już za sobą, a potem do wyniku dodaje pobrany numer dnia miesiąca.

W linii `(2)` korzystamy ze zmiennej `ileDniMinelo`, aby obliczyć ile dni zostało do końca roku. Na końcu metody `main` wyświetlamy komunikat dla użytkownika.

Przykładowe uruchomienie:

```
Podaj numer miesiąca: 1
Podaj numer dnia miesiąca: 1
Liczba dnia do końca roku: 359
```

```
Podaj numer miesiąca: 3
Podaj numer dnia miesiąca: 15
Liczba dnia do końca roku: 285
```

6.2.3. Różne rodzaje dzielenia

Napisz program, w którym pobierzesz od użytkownika dwie liczby. Następnie, program powinien wypisać na ekran wyniki podzielenia tych liczb na trzy sposoby:

- Wartości powinny być podzielone w taki sposób, aby wynikiem była liczba zmiennoprzecinkowa, tzn. posiadająca część ułamkową.
- Powinna zostać wyliczona reszta z dzielenia pierwszej liczby przez drugą.
- Wyliczona wartość dzielenia powinna zostać zaokrąglona do góry, tzn. jeżeli użytkownik poda liczby `10` i `4`, to wynikiem powinna być liczba `3.0` (`10` podzielone przez `4` to `2.5`, a `2.5` zaokrąglone do góry to `3.0`). Aby zaokrąglić liczbę do góry skorzystaj z metody `Math.ceil`. Przyjmuje ona jako argument jedną wartość typu `double` i zwraca ją zaokrągloną do góry. Ta zwrócona wartość także jest typu `double`.

Proponowane rozwiązanie wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_06/RozneRodzajeDzielenia.java

```
import java.util.Scanner;

public class RozneRodzajeDzielenia {
  public static void main(String[] args) {
    System.out.print("Podaj pierwszą liczbę: ");
    int x = pobierzLiczbe();

    System.out.print("Podaj drugą liczbę: ");
    int y = pobierzLiczbe();

    double wynikZmiennoprzecinkowy = (double) x / y; // 1
```

```
    int resztaZDzielenia = x % y; // 2

    double zaokragloneDzielenie =
        Math.ceil((double) x / y); // 3

    System.out.println(
        "Wynik dzielenia zmiennoprzecinkowego: " +
        wynikZmiennoprzecinkowy
    );

    System.out.println(
        "Reszta z dzielenia: " + resztaZDzielenia
    );

    System.out.println(
        "Wynik dzielenia zaokrąglony w górę: " +
        zaokragloneDzielenie
    );
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` pobieramy od użytkownika dwie liczby. Następnie wykonujemy trzy rodzaje dzielenia, zgodnie z wymaganiami zadania. W linii `(1)` rzutujemy jedną z pobranych liczb, aby wykonane zostało dzielenie zmiennoprzecinkowe, tzn. zawierające część ułamkową:

```
double wynikZmiennoprzecinkowy = (double) x / y; // 1
```

W linii `(2)` obliczamy wynik reszty z dzielenia korzystając z operatora modulo `%`:

```
int resztaZDzielenia = x % y; // 2
```

W linii `(3)` ponownie wykonujemy dzielenie zmiennoprzecinkowe, a wynik przekazujemy do metody `Math.ceil`, która zaokrągla w górę przekazany jej argument i zwraca go:

```
double zaokragloneDzielenie =
    Math.ceil((double) x / y); // 3
```

Przykładowe uruchomienie tego programu:

```
Podaj pierwszą liczbę: 10
Podaj drugą liczbę: 4
Wynik dzielenia zmiennoprzecinkowego: 2.5
Reszta z dzielenia: 2
Wynik dzielenia zaokrąglony w górę: 3.0
```

INSTRUKCJE WARUNKOWE

7.1. 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");
}
```

Ten kod jest niepoprawny, ponieważ warunek instrukcji warunkowej powinien być ujęty w nawiasy `()`:

```
if (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");
}
```

Ten kod jest niepoprawny, ponieważ sekcja `else` instrukcji warunkowej powinna być zawsze na jej końcu. Poprawnie zapisany kod wygląda następująco:

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

```
int a = 5;

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

Ten fragment kodu nie jest poprawny, ponieważ wyrażenie w warunku instrukcji warunkowej nie ma jednej z dwóch wymaganych wartości: `true` / `false`.

```
boolean b = true;

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

Ten fragment kodu jest poprawny.

```
int y = 5;

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

Ten fragment kodu nie jest poprawny, ponieważ wyrażenie w warunku instrukcji warunkowej nie ma jednej z dwóch wymaganych wartości: `true` / `false`. Pojedynczy znak równa się = to operator przypisania, a nie operator porównujący dwie wartości, czyli operator ==.

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

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

Ten fragment kodu jest poprawny.

---

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.");
}
```

Dla zmiennej `x` mającej wartość `10`, na ekranie zobaczymy:

```
10
20
```

Spowodowane jest to tym, że po dopasowaniu wartości z sekcji `case` wykonywane są wszystkie kolejne sekcje, dopóki nie napotkane zostanie słowo kluczowe `break`. Tego słowa nie ma w sekcji `case 10`, więc wykonane zostaną także instrukcje przyporządkowane do sekcji `case 20`. W tej sekcji jest instrukcja `break`, więc działanie instrukcji `switch` zakończy się w tym miejscu.

Dla zmiennej `x` mającej wartość `20`, na ekranie zobaczymy:

```
20
```

Dla zmiennej `x` mającej wartość `30`, na ekranie zobaczymy:

```
30
Inna wartość.
```

Podobnie jak w przypadku, gdy zmienna `x` ma wartość `10`, wykonane zostaną nie tylko instrukcje skojarzone z sekcją `case 30`, ale także z sekcją `default`, ponieważ w sekcji `case 30` zabrakło użycia słowa kluczowego `break`.

Dla zmiennej `x` mającej np. wartość `100` na ekranie zobaczymy:

```
Inna wartość.
```

3. Jaki będzie wynik działania poniższego programu?

```
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.");
}
```

Ten kod się nie skompiluje, ponieważ w instrukcjach `switch` nie można używać wartości typu `double`.

---

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;
```

Nie, instrukcja `if` i użycie trójargumentowego operatora warunkowego w tym przypadku nie są sobie równoważne. Instrukcja `if` jest poprawnie użyta do wyznaczenia wartości absolutnej zmiennej `z`, natomiast w przypadku trójargumentowego operatora warunkowego wyrażenia dla prawdy/fałszu są odwrócone. Fragment kodu z trójargumentowym operatorem warunkowym powinien być zapisany następująco:

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

---

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

Aby porównać treść dwóch wartości typu `String`, tzn. poznać odpowiedź na pytanie: „Czy te wartości składają się z takich samych znaków w tej samej kolejności?” należy skorzystać z metody `equals`.

Jednakże, jeżeli chcielibyśmy odpowiedzieć na pytanie, czy dwie zmienne typu `String` wskazują na ten sam obiekt w pamięci, moglibyśmy użyć porównania za pomocą operatora ==, jednak tego typu porównania zazwyczaj świadczą o błędzie w kodzie programu.

---

6. W jaki sposób można skrócić poniższy kod?

```
int x = 5;
boolean czyDodatnia;

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

Instrukcja warunkowa jest w tym przypadku nadmiarowa, ponieważ możemy do zmiennej `czyDodatnia` bezpośrednio przypisać wynik porównania `x > 0`:

```
czyDodatnia = x > 0;
```

---

7. Jaka będzie wartość zmiennej `pewnaZmienna`?

```
boolean pewnaZmienna;

pewnaZmienna = !pewnaZmienna;
```

Ten kod się nie skompiluje, ponieważ nie nadaliśmy zmiennej `pewnaZmienna` żadnej wartości – nie możemy jej więc użyć do wyznaczenia jej własnej wartości. Kompilator zwróci błąd „variable pewnaZmienna might not have been initialized”.

---

8. Czy operatora = można używać do porównywania wartości?

Nie, operator = to operator przypisania – do porównywania wartości używamy operatora relacyjnego ==.

---

9. Jaka będzie wartość poniższego wyrażenia, jeżeli `x = -5`, `y = -10`, `z = 20` ?

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

Wartość tego wyrażenia to `true`. Operator `&&` ma wyższy priorytet, niż operator `||`. Gdyby było na odwrót, to wartością byłoby `false`. Ten warunek jest równoważny poniższemu:

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

---

10. Jakie wartości mogą mieć zmienne typu `boolean`?

Zmienne typu `boolean` mogą przyjmować jedną z dwóch wartości: `true` bądź `false`.

---

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?

Wartość wyrażenia `y > 0` nie musi być obliczona, jeżeli wartością wyrażenia `x > 0` będzie `false`, ponieważ w takim przypadku wartość całego wyrażenia `x > 0 && y > 0` nie ma szansy mieć wartości `true`. Operator `&&` zwraca `true` tylko, gdy jego oba argumenty mają wartość `true`. Funkcjonalność, która powoduje, że niektóre wyrażenia nie są obliczane, nazywa się *short-circuit evaluation* i jest dla nas dostępna automatycznie w Maszynie Wirtualnej Java.

---

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.");
}
```

Ta instrukcja `switch` używająca nowej składni wygląda następująco:

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

Ten kod się nie skompiluje, ponieważ zmienna `poleKwadratu` istnieje jedynie w instrukcji warunkowej `if`. Gdy ta instrukcja się kończy, zmienna `poleKwadratu` przestaje istnieć, więc nie możemy z niej skorzystać podczas wypisywania na ekran pola kwadratu.

---

14. Dla jakich wartości argumentów operatory `&&` oraz `||` zwracają `true`? A dla jakiego argumentu zwraca wartość `true` operator `!` (logiczne zaprzeczenie)?

Operator `&&` zwraca `true` tylko wtedy, gdy oba jego argumenty mają wartość `true`.

Operator `||` zwraca `true`, gdy chociaż jeden z jego argumentów ma wartość `true`.

Operator `!` zwraca `true` w przypadku, gdy jego argument ma wartość `false`.

---

15. Który z operatorów ma wyższy priorytet: `||` czy `&&` ?

Operator `&&` ma wyższy priorytet od operatora `||`.

---

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
);
```

Program zawsze wypisze wartość przeciwną niż podana przez użytkownika, np. dla `5` wypisze `-5`, a dla `-5` wypisze `5`. W tym fragmencie kodu z instrukcją warunkową skojarzona jest tylko jedna instrukcja – wypisująca na ekran komunikat „x jest mniejsze od 0”. Instrukcja `x = -x;` także powinna być częścią instrukcji skojarzonych

z instrukcją `if`, tzn. powinna być otoczona nawiasami klamrowymi razem z instrukcją wypisującą komunikat na ekran. Poprawiony kod wygląda następująco:

```
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.");
}
```

Wypisany zostanie komunikat „Słoneczna pogoda.”, ponieważ wielkość znaków podczas porównywania stringów ma znacznie – `"Będzie padać"` i `"będzie padać"` to dwa różne łańcuchy tekstowe.

---

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";
};
```

Ten fragment kodu się nie skompiluje – wyrażenie `switch` musi brać pod uwagę wszystkie możliwe wartości lub zawierać sekcję `default`. Kompilator zwróci błąd „the switch expression does not cover all possible input values”.

---

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);
```

Kod się nie skompiluje, ponieważ zmiennej `poleKwadratu` nadajemy wartość w instrukcji warunkowej `if` – istnieje szansa na takie wykonanie tego kodu, w którym `poleKwadratu` nie otrzyma wartości. Kompilator zaprotestuje, ponieważ zmienna `poleKwadratu` może nie mieć nadanej wartości zanim zostanie użyta w ostatniej linii.

---

20. Czym różni się instrukcja `switch` od wyrażenia `switch`?

Instrukcja `switch` wykonuje instrukcje, które są przyporządkowane do jej sekcji `case`, natomiast wyrażenie `switch` zwraca wartość z sekcji `case`. Ponadto, instrukcja `switch` nie musi brać pod uwagę wszystkich możliwych wartości podczas dopasowywania sprawdzanej wartości.

---

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.";
  }
};
```

W tej postaci ten fragment kodu się nie skompiluje. Jeżeli sekcja `case` wyrażenia `switch` ma skojarzoną więcej, niż jedną instrukcję, to należy w niej skorzystać ze słowa kluczowego `yield` aby wskazać wartość, którą chcemy z tej sekcji zwrócić. Ten fragment kodu po poprawie wygląda następująco:

```
int dzienTygodnia = 1;

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

Do zmiennej `rodzajDnia` przypisana zostanie wartość `"Dzień roboczy."`.

7.2. Zadania

7.2.1. 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.

Rozwiązanie korzystające z instrukcji warunkowej wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_07/NajwiekszaZCzterechLiczbV1.java

```
import java.util.Scanner;

public class NajwiekszaZCzterechLiczbV1 {
  public static void main(String[] args) {
    System.out.println("Podaj cztery liczby:");
    int a = pobierzLiczbe();
    int b = pobierzLiczbe();
    int c = pobierzLiczbe();
    int d = pobierzLiczbe();

    if (a > b && a > c && a > d) {
      System.out.println("Największa liczba to: " + a);
    } else if (b > c && b > d) {
      System.out.println("Największa liczba to: " + b);
    } else if (c > d) {
      System.out.println("Największa liczba to: " + c);
    } else {
      System.out.println("Największa liczba to: " + d);
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` pobieramy cztery liczby. W instrukcji warunkowej sprawdzamy w pierwszej kolejności, czy to pierwsza z podanych liczb jest największa:

```
if (a > b && a > c && a > d) {
```

Jeżeli liczba zapisana w zmiennej `a` jest większa od pozostałych, to wypisujemy stosowny komunikat. Gdy ten warunek nie będzie spełniony, będzie to oznaczało, że któraś z pozostałych liczb jest większa, niż `a`. W takim przypadku przejdziemy do sprawdzenia, czy to liczba `b` jest największa:

```
} else if (b > c && b > d) {
```

W tym warunku nie ma już porównania do zmiennej `a`, ponieważ wiemy, że na pewno jest mniejsza od którejś z pozostałych liczb. Kontynuujemy w ten sposób dla liczby zapisanej w zmiennej `c`. Jeżeli ta liczba nie będzie większa od liczby `d`, to przejdziemy do sekcji `else` – będzie to oznaczało, że ostatnia liczba jest największa.

Przykładowe uruchomienie tego programu:

```
Podaj cztery liczby:
100
25
0
-5
Największa liczba to: 100
```

Druga wersja tego programu korzysta z metody `Math.max`, która przyjmuje dwie liczby i zwraca największą z nich:

Pytania_i_zadania_rozwiazania/Rozdzial_07/NajwiekszaZCzterechLiczbV2.java

```
import java.util.Scanner;

public class NajwiekszaZCzterechLiczbV2 {
  public static void main(String[] args) {
    System.out.println("Podaj cztery liczby:");
    int a = pobierzLiczbe();
    int b = pobierzLiczbe();
    int c = pobierzLiczbe();
    int d = pobierzLiczbe();

    System.out.println(
        "Największa liczba to: " +
        Math.max(Math.max(a, b), Math.max(c, d))
    );
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Ta wersja programu jest krótsza, ponieważ oddelegowaliśmy znalezienie największej liczby metodzie `Math.max`. Korzystamy z niej trzy razy – najpierw wyznaczmy największą liczbę z dwóch par liczb: `a` oraz `b` i `c` oraz `d`. Wynik, czyli dwie największe liczby z tych par, ponownie przekazuje jako argumenty do metody `Math.max`. W ten sposób otrzymamy w wyniku największą spośród wszystkich czterech liczb. Przykładowe uruchomienie tej wersji rozwiązania:

```
Podaj cztery liczby:
10
20
5
0
Największa liczba to: 20
```

7.2.2. 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.

Zwróć uwagę, że w proponowanym rozwiązaniu pobrane imię porównujemy za pomocą metody `equals`, a nie operatora `==`:

Pytania_i_zadania_rozwiazania/Rozdzial_07/SprawdzImie.java

```
import java.util.Scanner;

public class SprawdzImie {
  public static void main(String[] args) {
    System.out.print("Podaj swoje imię: ");
    String imie = pobierzSlowo();

    if (imie.equals("Przemek")) {
      System.out.println("Mamy takie samie imiona.");
    } else {
      System.out.println("Mamy różne imiona.");
    }
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

Dwa przykładowe uruchomienia tego programu:

```
Podaj swoje imię: Adam
Mamy różne imiona.
```

```
Podaj swoje imię: Przemek
Mamy takie samie imiona.
```

7.2.3. 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.

Proponowane rozwiązania tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_07/CzyOsobaPelnoletnia.java

```
import java.util.Scanner;

public class CzyOsobaPelnoletnia {
  public static void main(String[] args) {
    System.out.print("Podaj swój wiek: ");
    int wiek = pobierzLiczbe();

    boolean czyOsobaPelnoletnia = wiek >= 18;

    if (czyOsobaPelnoletnia) {
      System.out.println("Jesteś osobą pełnoletnią.");
    } else {
      System.out.println("Nie jesteś osobą pełnoletnią.");
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Po pobraniu od użytkownika wieku do zmiennej `czyOsobaPelnoletnia` przypisujemy bezpośrednio wynik porównania wieku osoby do liczby `18`. Możemy wykonać przypisanie w ten sposób, ponieważ operator relacyjny `>=` zwraca jedną z dwóch

wartości: `true` lub `false`, czyli jedną z wartości, jakie mogą przechowywać zmienne typu `boolean`. Na końcu programu wypisujemy informacje w zależności od tego, czy podano wiek osoby pełnoletniej, czy nie. Dwa przykładowe uruchomienia:

```
Podaj swój wiek: 18
Jesteś osobą pełnoletnią.
```

```
Podaj swój wiek: 15
Nie jesteś osobą pełnoletnią.
```

7.2.4. 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.

Przykładowe rozwiązanie tego zadania wygląda następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_07/MalejacyLubRosnacyCiagTrzechLiczb.java

```
import java.util.Scanner;

public class MalejacyLubRosnacyCiagTrzechLiczb {
  public static void main(String[] args) {
    System.out.println("Podaj trzy liczby:");
    int x = pobierzLiczbe();
    int y = pobierzLiczbe();
    int z = pobierzLiczbe();

    if (x > y && y > z) { // 1
      System.out.println("Ten ciąg liczb jest malejący.");
    } else if (x < y && y < z) { // 2
      System.out.println("Ten ciąg liczb jest rosnący.");
    } else {
      System.out.println(
        "Nie jest to ani malejący ciąg liczb, ani rosnący."
      );
    }
  }
```

```
  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku tego programu pobieramy od użytkownika trzy liczby. W linii `(1)` sprawdzamy, czy kolejne liczby są coraz mniejsze – jeżeli tak, to świadczy to o tym, że ten ciąg liczb jest malejący. W linii `(2)` sprawdzamy natomiast, czy liczby są coraz większe, czyli czy stanowią ciąg rosnący. W sekcji `else` obsługujemy sytuację, w której podane przez użytkownika liczby nie stanowią ani ciągu malejącego, ani rosnącego. Kilka przykładowych uruchomień tego programu:

```
Podaj trzy liczby:
5
10
15
Ten ciąg liczb jest rosnący.
```

```
Podaj trzy liczby:
5
5
10
Nie jest to ani malejący ciąg liczb, ani rosnący.
```

```
Podaj trzy liczby:
3
2
1
Ten ciąg liczb jest malejący.
```

7.2.5. 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.

Rozwiązanie do tego zadania może wyglądać następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_07/LiczeniePolaFigury.java

```
import java.util.Scanner;

public class LiczeniePolaFigury {
  public static void main(String[] args) {
    System.out.println("""
        Podaj nazwę figury do policzenia pola:
          kolo
          kwadrat
          prostokat
        """
    ); // 1
    System.out.print("Którą figurę wybierasz? ");
    String nazwaFigury = pobierzSlowo(); // 2

    if ("kolo".equalsIgnoreCase(nazwaFigury)) { // 3
      System.out.print("Podaj promień tego koła: ");
      int promien = pobierzLiczbe();

      if (promien <= 0) { // 4
        System.out.println("Nieprawidłowy promień koła.");
      } else {
        System.out.printf(
            "Promień koła o promieniu %d wynosi %.2f.",
            promien,
            Math.PI * promien * promien
        );
      }
    } else if ("kwadrat".equalsIgnoreCase(nazwaFigury)) {
      System.out.print("Podaj bok tego kwadratu: ");
      int bokKwadratu = pobierzLiczbe();

      if (bokKwadratu <= 0) { // 5
        System.out.println("Nieprawidłowy bok kwadratu.");
      } else {
        System.out.printf(
            "Pole kwadratu o boku %d wynosi %d.",
            bokKwadratu,
            bokKwadratu * bokKwadratu
        );
      }
    } else if ("prostokat".equalsIgnoreCase(nazwaFigury)) {
      System.out.print(
          "Podaj pierwszy bok tego prostokąta: "
```

```
      );
      int bokA = pobierzLiczbe();

      System.out.print(
          "Podaj drugi bok tego prostokąta: "
      );
      int bokB = pobierzLiczbe();

      if (bokA <= 0 || bokB <= 0) { // 6
        System.out.println(
            "Nieprawidłowe boki prostokąta."
        );
      } else {
        System.out.printf(
            "Pole prostokąta o bokach %d i %d wynosi %d",
            bokA,
            bokB,
            bokA * bokB
        );
      }
    } else { // 7
      System.out.println(
          "Nieznana nazwa figury: " + nazwaFigury
      );
    }
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Na początku metody `main` `(1)` informujemy użytkownika jakich figur pole możemy dla niego policzyć. W linii `(2)` pobieramy nazwę wybranej figury. Następnie, w instrukcji warunkowej, która zaczyna się w linii `(3)`, sprawdzamy, która figura została wybrana. Do porównania nazwy figury korzystamy z metody `equalsIgnoreCase` typu `String`, dzięki czemu niezależnie od tego, czy użytkownik wpisze nazwę figury małymi czy wielkimi literami, to i tak zostanie ona poprawnie rozpoznana.

W zależności od tego pobieramy od użytkownika odpowiednią wartość – będzie to długość promienia koła albo długość boku kwadratu bądź długości boków prostokąta. W każdej sekcji instrukcji warunkowej `if` sprawdzamy, czy użytkownik podał

poprawne dane (linie `(4)`, `(5)`, `(6)`). Jeżeli nie, to wyświetlamy stosowny komunikat, a w przeciwnym razie wypisujemy pole dla danej figury.

Jeżeli podana przez użytkownika nazwa figury nie zostanie dopasowana, to w sekcji `else` z linii `(7)` poinformujemy o tym fakcie użytkownika.

Kilka przykładowych uruchomień tego programu:

```
Podaj nazwę figury do policzenia pola:
  kolo
  kwadrat
  prostokat

Którą figurę wybierasz? trapez
Nieznana nazwa figury: trapez
```

```
Podaj nazwę figury do policzenia pola:
  kolo
  kwadrat
  prostokat

Którą figurę wybierasz? kolo
Podaj promień tego koła: 5
Promień koła o promieniu 5 wynosi 78,54.
```

```
Podaj nazwę figury do policzenia pola:
  kolo
  kwadrat
  prostokat

Którą figurę wybierasz? kwadrat
Podaj bok tego kwadratu: 5
Pole kwadratu o boku 5 wynosi 25.
```

```
Podaj nazwę figury do policzenia pola:
  kolo
  kwadrat
  prostokat

Którą figurę wybierasz? prostokat
Podaj pierwszy bok tego prostokąta: 5
Podaj drugi bok tego prostokąta: 0
Nieprawidłowe boki prostokąta.
```

7.2.6. 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;
```

Proponowane rozwiązanie wygląda jak poniżej. Ze względu na użycie w tym programie klasy `ThreadLocalRandom` do losowania liczb, na początku programu dodałem do niego instrukcję `import` (linia `(1)`).

Na początku metody `main` losowane są dwie liczby z przedziału od 0 do 19 (włącznie) – tak działa metoda `nextInt` – zwraca ona wartość od 0 do podanej wartości minus jeden. Dlatego do wylosowanej wartości dodawana jest liczba 1 – dzięki temu wynikiem będzie liczba z przedziału od 1 do 20.

W linii `(2)` wyświetlany jest komunikat dla użytkownika, w którym zawarta jest informacja o wylosowanych liczbach.

Pytania_i_zadania_rozwiazania/Rozdzial_07/SprawdzanieMnozenia.java

```
import java.util.Scanner;
import java.util.concurrent.ThreadLocalRandom; // 1

public class SprawdzanieMnozenia {
  public static void main(String[] args) {
    int liczba1 =
        ThreadLocalRandom.current().nextInt(20) + 1;

    int liczba2 =
        ThreadLocalRandom.current().nextInt(20) + 1;

    System.out.printf("""
        Wylosowałem liczby %d i %d. \
        Ile wynosi ich iloczyn?
        """,
        liczba1, liczba2
    ); // 2
```

```
    System.out.print("Podaj wynik: ");
    int wynikOdUzytkownika = pobierzLiczbe();
    int wynikMnozenia = liczba1 * liczba2;

    if (wynikOdUzytkownika == wynikMnozenia) { // 3
      System.out.println("To poprawna odpowiedź!");
    } else {
      System.out.printf(
          "Błędna odpowiedź - ten iloczyn to %d.",
          wynikMnozenia
      );
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

W dalszej części programu pytamy użytkownika o wynik mnożenia tych dwóch liczb. W linii `(3)` porównujemy wynik podany przez użytkownika do wyniku obliczonego w programie. Jeżeli się zgadza, to informujemy o tym fakcie użytkownika, a w przeciwnym razie wypisujemy poprawny wynik.

Dwa przykładowe uruchomienia tego programu:

```
Wylosowałem liczby 10 i 6. Ile wynosi ich iloczyn?
Podaj wynik: 60
To poprawna odpowiedź!
```

```
Wylosowałem liczby 12 i 11. Ile wynosi ich iloczyn?
Podaj wynik: 121
Błędna odpowiedź - ten iloczyn to 132.
```

7.2.7. 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.
```

Do rozwiązania tego zadania posłuży nam metoda `indexOf` typu `String` przyjmująca string do znalezienia w łańcuchu tekstowym, na rzecz którego została wywołana. Jeżeli zostanie on znaleziony, to `indexOf` zwróci indeks, od którego ten string się zaczyna, a w przeciwnym razie liczbę `-1`. Kilka przykładów:

```
String slowo = "programowanie";

// wypisze 3 - string "gram" znajduje się w słowie
// "programowanie" i zaczyna się od trzeciego indeksu
// (pamiętaj, że indeksy zaczynamy numerować od 0, a nie 1)
System.out.println(slowo.indexOf("gram"));

// wypisze -1 - string "nauka" nie występuje
// w słowie "programowanie"
System.out.println(slowo.indexOf("nauka"));
```

Opis metody `indexOf` w oficjalnej dokumentacji:

https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/String.html#indexOf(java.lang.String)

Spójrzmy na proponowane rozwiązanie. Na początku poniższego programu pobieramy od użytkownika nazwę zapisaną w standardzie snake_case. W linii `(1)` korzystamy z metody `indexOf`, wywołując ją na pobranej nazwie, a jako argument przekazując znak podkreślenia. Dzięki temu dowiemy się w którym miejscu w tej nazwie znajduje się znak oddzielający słowa.

Pytania_i_zadania_rozwiazania/Rozdzial_07/SnakeCaseNaCamelCase.java

```
import java.util.Scanner;

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

```
    System.out.println("""
        Podaj nazwę składającą się z dwóch słów \
        zapisaną w standardzie snake_case:"""
    );
    String nazwaWSnakeCase = pobierzSlowo();

    // 1
    int indeksPodkreslenia = nazwaWSnakeCase.indexOf("_");

    if (indeksPodkreslenia == -1) { // 2
      System.out.println(
          "W podanej nazwie nie ma znaku podkreślenia."
      );
    } else {
      String pierwszeSlowo =
          nazwaWSnakeCase.substring(0, indeksPodkreslenia);

      String drugieSlowo =
          nazwaWSnakeCase.substring(indeksPodkreslenia + 1);

      if ( // 3
          pierwszeSlowo.length() == 0 ||
          drugieSlowo.length() == 0
      ) {
        System.out.println(
            "Podana nazwa nie składa się z dwóch słów."
        );
      } else { // 4
        String nazwaWCamelCase =
            pierwszeSlowo.toLowerCase() +
            Character.toUpperCase(drugieSlowo.charAt(0)) +
            drugieSlowo.substring(1).toLowerCase();

        System.out.println(
            "Ta nazwa zapisana w standardzie Camel Case to " +
            nazwaWCamelCase
        );
      }
    }
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

Jeżeli wartością zmiennej `indeksZnakuPodkreslenia` będzie `-1`, co sprawdzamy w linii `(2)`, będzie to oznaczać, że w podanej przez użytkownika nazwie nie ma znaku podkreślenia – wypiszemy więc stosowny komunikat.

W przeciwnym razie, w sekcji `else`, skorzystamy dwukrotnie z metody `substring`, aby pobrać pierwsze i drugie słowo z nazwy. Jako wyznacznik końca pierwszego słowa i początek drugiego posłuży nam indeks znaku podkreślenia:

```
String pierwszeSlowo =
    nazwaWSnakeCase.substring(0, indeksPodkreslenia);

String drugieSlowo =
    nazwaWSnakeCase.substring(indeksPodkreslenia + 1);
```

Pierwsze użycie `substring` zwróci znaki od początku nazwy aż do znaku poprzedzającego znak podkreślenia (ostatnim znakiem zwracanym przez tę wersję metody `substring` jest znak *przed* indeksem podanym jako drugi argument). Drugie wywołanie metody `substring` zwróci fragment zaczynający się zaraz za znakiem podkreślenia i zawierający wszystkie znajdujące się za nim znaki.

Istnieje szansa, że użytkownik poda nazwę, która zaczyna się od znaku podkreślenia lub nim kończy, np. `_nazwa` lub `nazwa_`. W takim przypadku nazwa składa się z tylko jednego słowa, więc zmienna `pierwszeSlowo` lub `drugieSlowo` będzie pustym stringiem – sprawdzamy te przypadki w linii `(3)` i wypisujemy ewentualny komunikat.

Jeżeli użytkownik poda poprawną nazwę, to w sekcji `else` z linii `(4)` utworzymy z obu słów nazwę w standardzie Camel Case i wypiszemy ją na ekran.

Przykładowe uruchomienie tego programu:

```
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_
Podana nazwa nie składa się z dwóch słów.
```

```
Podaj nazwę składającą się z dwóch słów zapisaną w standardzie snake_case:

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.
```

7.2.8. 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 zadaniu „7.2.6 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.
```

W rozwiązaniu tego zadania korzystamy z klasy `ThreadLocalRandom`, dzięki której będzie mieli dostęp do metody losującej liczby, jak i z klasy `Scanner`, która, tym razem, posłuży nam do pobrania od użytkownika całej linii tekstu. W związku z tym na początku programu znajdują się dwie instrukcji `import`.

Na początku metody `main` pobieramy od użytkownika linię tekstu – pytanie, na które chce poznać odpowiedź. Następnie, losujemy numer odpowiedzi – będzie to liczba z przedziału od `0` do `7` (tak działa metoda `nextInt`), więc do wyniku dodamy `1`, aby finalnie w zmiennej `numerOdpowiedzi` mieć liczbę z zakresu od `1` do `8`.

Pytania_i_zadania_rozwiazania/Rozdzial_07/MagicznaKulaNr8.java

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

public class MagicznaKulaNr8 {
  public static void main(String[] args) {
    System.out.println("Co chcesz wiedzieć?");
    String pytanie = pobierzLinieTekstu();

    int numerOdpowiedzi =
        ThreadLocalRandom.current().nextInt(8) + 1;
```

```
    String odpowiedz = switch (numerOdpowiedzi) { // 1
      case 1 -> "Tak.";
      case 2 -> "Być może.";
      case 3 -> "Raczej nie.";
      case 4 -> "Nie.";
      case 5 -> "Nie wiem.";
      case 6 -> "Nie liczyłbym na to.";
      case 7 -> "Lepiej, żebyś nie wiedział(a).";
      default ->
          "W tej chwili Ci nie powiem, spróbuj później.";
    };

    System.out.printf(
        "Pytasz \"%s\"... %s", pytanie, odpowiedz
    );
  }

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

Wylosowanej wartości używamy do wyznaczenia odpowiedzi dla użytkownika. W tym celu skorzystałem z wyrażenia `switch`, które zaczyna się w linii `(1)`. Wyrażenie `switch` musi albo brać pod uwagę wszystkie możliwe wartości, albo posiadać sekcję `default`, dlatego obsługę wylosowania odpowiedzi numer `8` zapisałem w sekcji `default`. Na końcu metody `main` wypisujemy komunikat dla użytkownika.

Przykładowe uruchomienie tego programu:

```
Co chcesz wiedzieć?
Czy na urlopie będzie padać?
Pytasz "Czy na urlopie będzie padać?"... Być może.
```

PĘTLE

8.1. Pytania

1. Do czego służą pętle?

Pętle służą do wielokrotnego wykonywania zestawu instrukcji, który nazywamy *ciałem* pętli. Pętla wykonuje się tak długo, jak jej warunek jest spełniony, tzn. ma wartość `true`.

---

2. Czym różnią się od siebie poznane dotąd pętle?

Instrukcje zawarte w ciałach pętli `while` oraz `for` mogą nie wykonać się ani razu, jeżeli warunek pętli nie będzie spełniony na początku działania pętli, w przeciwieństwie do pętli `do…while`, której instrukcje wykonają się co najmniej raz.

Pętla `for` jako jedyna posiada trzy człony – instrukcję inicjalizującą, warunek, oraz instrukcję kroku. Pozostałe dwie pętle posiadają jedynie warunek pętli.

---

3. Jak nazywamy obieg pętli?

Obieg pętli nazywamy *iteracją*.

---

4. Czy ciało pętli zawsze wykona się chociaż raz?

Zależy to od rodzaju pętli, którego użyjemy. Ciało pętli `do…while` zawsze wykona się przynajmniej raz, ponieważ jej warunek sprawdzany jest na końcu iteracji. Z kolei pętle `for` i `while` sprawdzają swój warunek na początku, więc ich ciała mogą się nie wykonać ani razu.

---

5. Co się stanie, gdy warunek pętli będzie zawsze spełniony i nie zmieni się w trakcie działania programu?

Program będzie działał „w nieskończoność”. Jeżeli jest to program konsolowy (uruchamiany w linii poleceń), to możemy spróbować zakończyć jego działanie skrótem Ctrl + C.

---

6. Z jakich części składa się pętla `for`? Czy są one wymagane?

Pętla `for` posiada: instrukcję inicjalizującą, warunek, oraz instrukcję kroku, a także ciało pętli. Żadna z nich nie jest wymagana.

---

7. Jak sprawdzić znak na danej pozycji w zmiennej typu `String`? Jak sprawdzić pierwszy znak, a jak ostatni?

Aby otrzymać znak na danej pozycji w zmiennej typu `String`, korzystamy z metody `charAt`, której przekazujemy jako argument indeks znaku, który chcemy pobrać. Indeksy znaków zaczynają się od `0`, a nie `1`, więc ostatni znak ma indeks o jeden mniejszy niż liczba znaków w stringu. Możemy się do niego odnieść korzystając z metody `length`, która zwraca liczbę znaków w stringu:

```
char osatatniZnak = tekst.charAt(tekst.length() - 1);
```

---

8. Czy poniższe fragmenty kodów źródłowych są sobie równoważne?

```
int i = 0;
while (i < 10) {
  System.out.print(i + " ");
  i++;
}
```

```
for (int i = 0; i < 10; i++) {
  System.out.print(i + " ");
}
```

Nie, te fragmentu kodu nie są sobie równoważne, ponieważ w przypadku pętli `for` zmienna `i` nie jest dostępna po zakończeniu działania pętli. W przypadku pierwszego fragmentu (z pętlą `while`) moglibyśmy nadal korzystać ze zmiennej `i`, ponieważ została zdefiniowana jeszcze przed pętlą, a nie wewnątrz niej, jak w przypadku fragmentu z pętlą `for`.

---

9. Do czego służą instrukcje `break` oraz `continue`?

Instrukcja `break` służy do natychmiastowego przerwania działania pętli, w której została użyta, natomiast instrukcja `continue` przerywa aktualną iterację pętli.

---

10. Jaki będzie wynik działania poniższego fragmentu kodu?

```
for (int i = 0; i < 3; i++) {
  for (int j = 0; j < 10; j++) {
    System.out.println("j = " + j);

      if (j == 1) {
        break;
      }
  }
}
```

Instrukcja `break` w pętli wewnętrznej przerwie jedynie działanie tej wewnętrznej pętli – nie będzie ona miała wpływu na pętlę zewnętrzną, więc na ekranie zobaczymy:

```
j = 0
j = 1
j = 0
j = 1
j = 0
j = 1
```

---

11. Co zostanie wypisane w ramach działania poniższego fragmentu kodu?

```
String komunikat = "Witaj";

for (int i = 0; i <= komunikat.length(); i++) {
  System.out.print(komunikat.charAt(i) + " ");
}
```

Na ekranie zobaczymy komunikat `W i t a j` (ze spacjami po każdym znaku), a także komunikat błędu spowodowanym tym, że wyszliśmy poza zakres indeksów zmiennej `komunikat`. W pętli powinniśmy użyć operatora < zamiast <=, ponieważ indeks ostatniego znaku w łańcuchach tekstowych to `length() - 1`, a nie `length()`:

```
W i t a j Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
Index 5 out of bounds for length 5
  at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:55)
  at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:52)
  at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:213)
  at java.base/jdk.internal.util.Preconditions$4.apply(Preconditions.java:210)
  at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:98)
  at     java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Precondi-
tions.java:106)
  at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:302)
  at java.base/java.lang.String.checkIndex(String.java:4881)
  at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:46)
  at java.base/java.lang.String.charAt(String.java:1582)
  at WypiszWitaj.main(WypiszWitaj.java:6)
```

12. Zapisz kod poniżej tak, aby używał pętli `while`:

```
for (int i = 1, j = 1; i * j < 100; i++, j += 2) {
  System.out.print((i * j) + " ");
}
```

Podobny fragment kodu, korzystający z pętli `while`, wygląda następująco:

```
int i = 1, j = 1;

while (i * j < 100) {
  System.out.print((i * j) + " ");
  i++;
  j += 2;
}
```

Należy tutaj zwrócić uwagę, że te dwa fragmenty kodu nie są sobie równoważne – w przypadku pętli `for` po jej zakończeniu zmienne `i` oraz `j` przestają istnieć. We fragmencie z pętlą `while` te zmienne są zdefiniowane przed pętlą, więc istnieją także po zakończeniu pętli.

13. Co zostanie wypisane na ekranie po uruchomieniu poniższego fragmentu kodu?

```
int suma = 0;
for (int i = 1; i < 5; i++) {
  suma += i;
}
```

```
System.out.println("Suma ma wartość: " + suma);
System.out.println(
    "i po zakończeniu pętli ma wartość: " + i
);
```

Ten fragment kodu się nie skompiluje, ponieważ zmienna `i` istnieje jedynie w pętli, w której została zdefiniowana. Po zakończeniu pętli `for` zmienna `i` przestaje istnieć.

8.2. Zadania

8.2.1. While i liczby od 1 do 10

Napisz program z pętlą `while`, który wypisuje wszystkie liczby od `1` do `10` (włącznie), oddzielone przecinkami, poza liczbą `10`, po której nie powinno być przecinka.

W ciele pętli musimy sprawdzić wartość zmiennej `i` – gdy będzie równa `10`, nie wypiszemy na ekran przecinka:

Pytania_i_zadania_rozwiazania/Rozdzial_08/WhileILiczbyOd1Do10.java

```
public class WhileILiczbyOd1Do10 {
  public static void main(String[] args) {
    int i = 1;

    while (i <= 10) {
      if (i != 10) {
        System.out.print(i++ + ", ");
      } else {
        System.out.print(i++);
      }
    }
  }
}
```

Efekt uruchomienia tego programu:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

8.2.2. Policz silnię

Napisz program, który policzy i wypisze silnię liczby, którą poda użytkownik. Silnia to iloczyn kolejnych liczb naturalnych od `1` do danej liczby, np. silnia `5` to `1 * 2 * 3 * 4 * 5` czyli `120`. Silnia liczby `0` to `1`. Jeżeli podana przez użytkownika liczba będzie mniejsza niż `0`, program powinien wypisać informację, że oczekiwał liczby większej bądź równej zero, po czym powinien zakończyć działanie.

W poniższym przykładowym rozwiązaniu sprawdzamy najpierw pobraną liczbę – silnię powinniśmy liczyć jedynie dla liczb nieujemnych. Jeżeli liczba jest nieujemna, to w pętli `for` przemnażamy zmienną `wynik` przez kolejne liczby całkowite.

Zauważ, że podanie przez użytkownika liczby `0` spowoduje, że ciało pętli nie wykona się ani razu, ponieważ warunek pętli nie będzie spełniony. Wtedy zmienna `wynik` będzie miała po prostu swoją wartość nadaną jej podczas inicjalizacji, czyli `1`, a silnia `0` wynosi właśnie `1`.

Pytania_i_zadania_rozwiazania/Rozdzial_08/PoliczSilnie.java

```
import java.util.Scanner;

public class PoliczSilnie {
  public static void main(String[] args) {
    System.out.print("Podaj liczbę nieujemną: ");
    int liczba = pobierzLiczbe();

    if (liczba < 0) {
      System.out.println("Oczekiwałem liczby nieujemnej.");
    } else {
      int silnia = 1;

      for (int i = 1; i <= liczba; i++) {
        silnia = silnia * i;
      }

      System.out.printf(
          "Silnia liczby %d to %d.", liczba, silnia
      );
    }
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Trzy przykładowe uruchomienia tego programu:

```
Podaj liczbę nieujemną: -1
Oczekiwałem liczby nieujemnej.
```

```
Podaj liczbę nieujemną: 0
Silnia liczby 0 to 1.
```

```
Podaj liczbę nieujemną: 10
Silnia liczby 10 to 3628800.
```

8.2.3. Zagnieżdżone pętle

Napisz program z dwoma pętlami – jedna z nich powinna być zagnieżdżona w drugiej. Każda z pętli powinna przechodzić przez liczby od `1` do `10` (włącznie) oraz:

- Pętla główna powinna pomijać swoje iteracje za pomocą instrukcji `continue`, gdy używana w niej zmienna ma wartość nieparzystą.
- Pętla zagnieżdżona powinna wypisywać wartość swojej zmiennej używając `System.out.print`. Po wypisanej wartości powinna zostać dodana spacja. Następnie, gdy zmienna pętli zagnieżdżonej będzie większa od zmiennej pętli głównej, pętla zagnieżdżona powinna wypisać na ekran znak nowej linii (`System.out.println();`) i przerwać swoje dalsze działanie.

Przykładowe rozwiązanie tego zadania wygląda jak poniżej. Zgodnie z pierwszym wymaganiem, czyli gdy zmienna pętli zewnętrznej jest nieparzysta, instrukcja `continue` powoduje przejście do kolejnego obiegu pętli zewnętrznej. To sprawdzenie wykonujemy w linii `(1)`.

W pętli wewnętrznej wypisujemy na ekran wartości zmiennej `j`. Następnie sprawdzamy w linii `(2)`, czy wartość zmiennej `j` przekroczyła wartość zmiennej używanej w pętli zewnętrznej, czyli zmiennej `i`. Jeżeli tak, to zgodnie z wymaganiami zadania wypisujemy na ekran pustą linię i przerywamy działanie wewnętrznej pętli za pomocą instrukcji `break`.

Pytania_i_zadania_rozwiazania/Rozdzial_08/ZagniezdzonePetle.java

```
public class ZagniezdzonePetle {
  public static void main(String[] args) {
    for (int i = 1; i <= 10; i++) {
      if (i % 2 == 1) { // 1
        continue;
      }

      for (int j = 1; j <= 10; j++) {
        System.out.print(j + " ");

        if (j > i) { // 2
          System.out.println();
          break;
        }
      }
    }
  }
}
```

Wynik uruchomienia tego programu jest następujący:

```
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

8.2.4. Kalkulator

Napisz program, który będzie pobierał od użytkownika liczby i działania do wykonania na nich. Program powinien wypisywać wynik po każdym działaniu. Możliwe działania to: `*` (mnożenie), `/` (dzielenie), `-` (odejmowanie), `+` (dodawanie).

Jeżeli podane zostanie inne działanie, lub podana zostanie liczba `0` jako dzielnik podczas dzielenia, program powinien wypisać stosowny komunikat i ponownie pobrać od użytkownika dane.

Na początku program powinien pobrać od użytkownika dwie liczby i działanie do wykonania na nich. Za każdym kolejnym razem, program powinien pobierać od użytkownika już tylko jedną liczbę i działanie, po czym powinien wykonać podane działanie na poprzednim wyniku i podanej liczbie.

Dla przykładu:

1. Program pobiera najpierw dwie liczby od użytkownika: `10` i `15`, oraz działanie: + (dodawanie).
2. Program dodaje do siebie liczby i wypisuje wynik `25` na ekran.
3. Program pyta, czy użytkownik chce wykonać kolejne działanie.
   a. Jeżeli nie, program kończy działanie.
   b. Jeżeli tak, to program pobiera jedną liczbę i działanie, np. `2` i `*` (mnożenie). Program mnoży poprzedni wynik przez nowo pobraną liczbę, czyli wykonuje działanie `25 * 2`, i wypisuje wynik `50` na ekran. Wracamy do punktu trzeciego i ponownie pytamy o chęć dalszych kalkulacji.

W tym programie:

- musimy odpowiednio obsłużyć szczególny przypadek – na samym początku działania programu należy pobrać od użytkownika dwie liczby, aby wykonać na nich działanie. Następnie, po wykonaniu obliczenia, zostanie nam wynik, więc ewentualne następne obliczenie będzie wymagało podania przez użytkownika już tylko jednej liczby – kolejne działanie wykonamy na poprzednim wyniku i tej nowo pobranej od użytkownika liczbie,
- chcemy wypisać na ekran wykonywane działanie tylko w przypadku, gdy obliczenie się udało (tzn. użytkownik nie podał `0` jako mianownik podczas dzielenia bądź nieznanego działania).

Przykładowe rozwiązanie tego zadania wygląda jak poniżej. Na początku metody `main` pobieramy od użytkownika pierwszą liczbę – zapisujemy ją w zmiennej `poprzedniWynik`, która będzie nam dalej służyła do przechowywania poprzedniego wynika działania.

W linii `(1)` definiujemy zmienną `czyBlednaOperacja`, której wartość będziemy ustawiać na `true` gdy użytkownik poda nieznane działanie do wykonania lub `0` jako dzielnik.

W pętli `do...while` definiujemy zmienną `nowyWynik`, do której przypiszemy wynik działania wykonanego na poprzednim wyniku i nowo pobranej liczbie. Dodatkowo, ustawiamy w linii `(3)` wartość zmiennej `czyBlednaOperacja` na `false` – w poprzednim obiegu do tej zmiennej mogła zostać przypisana wartość `true`, jeżeli użytkownik podał nieprawidłowe dane. W każdym kolejnym obiegu chcemy założyć, że nie wystąpił błąd, aby po ponownym pobraniu operacji do wykonania i liczby móc ponownie ustawić wartość tej zmiennej na `true`, jeżeli okaże się, że użytkownik kolejny raz podał nieprawidłowe dane.

W kolejnych liniach pobieramy działanie do wykonania i drugą liczbę.

Mając już wszystkie dane, w instrukcji `switch` wykonujemy wybrane działanie na wyniku poprzedniego działania i na nowo pobranej liczbie. Jeżeli działanie nie zostanie rozpoznane, to w sekcji `default` w linii `(5)` ustawimy wartość zmiennej `czyBlednaOperacja` na `true` i wypiszemy stosowny komunikat. W linii `(4)` obsługujemy dzielenie – zanim je wykonamy, sprawdzamy, czy dzielnikiem nie jest liczba `0`. Jeżeli tak, to ustawimy wartość zmiennej `czyBlednaOperacja` na `true` i wypiszemy komunikat dla użytkownika, że nie można dzielić przez `0`.

Pytania_i_zadania_rozwiazania/Rozdzial_08/Kalkulator.java

```
import java.util.Scanner;

public class Kalkulator {
  public static void main(String[] args) {
    System.out.print("Podaj liczbę: ");
    int poprzedniWynik = pobierzLiczbe();

    boolean czyBlednaOperacja; // 1

    do {
      int nowyWynik = 0;
      czyBlednaOperacja = false; // 3

      System.out.print("Podaj działanie (* / - +): ");
      String dzialanie = pobierzSlowo();

      System.out.print("Podaj kolejną liczbę: ");
      int drugaLiczba = pobierzLiczbe();
```

```
      switch (dzialanie) {
        case "+" -> nowyWynik = poprzedniWynik + drugaLiczba;
        case "-" -> nowyWynik = poprzedniWynik - drugaLiczba;
        case "*" -> nowyWynik = poprzedniWynik * drugaLiczba;
        case "/" -> { // 4
          if (drugaLiczba == 0) {
            czyBlednaOperacja = true;
            System.out.println("Nie można dzielić przez 0.");
          } else {
            nowyWynik = poprzedniWynik / drugaLiczba;
          }
        }
        default -> { // 5
          czyBlednaOperacja = true;
          System.out.println(
              "Nieprawidłowa operacja: " + dzialanie
          );
        }
      }

      if (!czyBlednaOperacja) { // 6
        System.out.printf(
            "%d %s %d = %d\n",
            poprzedniWynik,
            dzialanie,
            drugaLiczba,
            nowyWynik
        );

        poprzedniWynik = nowyWynik; // 7
      }

      System.out.print(
          "\nCzy chcesz zakończyć program? [t/n] "
      );
    } while (!"t".equalsIgnoreCase(pobierzSlowo()));
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

W linii (6) wypisujemy wynik wykonanego działania tylko w przypadku, gdy nie napotkane zostały żadne błędy (nieznane działanie bądź dzielnik wynoszący 0). Jeżeli dane były poprawne, to w linii (7) zapisujemy aktualny wynik działania w zmiennej `poprzedniWynik` – w ten sposób obliczona wartość stanie się jedną z liczb, na których zostanie wykonane kolejne działanie. Na końcu pętli pytamy użytkownika, czy chce zakończyć program. Przykładowe uruchomienie tego programu:

```
Podaj liczbę: 10
Podaj działanie (* / - +): ^
Podaj kolejną liczbę: 5
Nieprawidłowa operacja: ^

Czy chcesz zakończyć program? [t/n] n
Podaj działanie (* / - +): *
Podaj kolejną liczbę: 5
10 * 5 = 50

Czy chcesz zakończyć program? [t/n] n
Podaj działanie (* / - +): +
Podaj kolejną liczbę: 25
50 + 25 = 75

Czy chcesz zakończyć program? [t/n] n
Podaj działanie (* / - +): /
Podaj kolejną liczbę: 0
Nie można dzielić przez 0.

Czy chcesz zakończyć program? [t/n] t
```

8.2.5. Choinka

Napisz program, który pobierze od użytkownika jedną liczbę całkowitą. Następnie, program powinien wypisać na ekran choinkę ze znaków `*`, gdzie w ostatniej linii będzie liczba gwiazdek podana przez użytkownika, a w każdej kolejnej linii o dwie gwiazdki mniej, niż w poprzedniej.

Przykład dla nieparzystej liczby gwiazdek – użytkownik podał liczbę 5, efekt wyświetlony na ekranie:

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

Przykład dla parzystej liczby gwiazdek – użytkownik podał liczbę 6:

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

Zauważ, że niezależnie od liczby gwiazdek, jaką podamy, na szczycie choinki (tzn. w pierwszym rzędzie) zawsze będzie albo jedna gwiazdka, albo dwie. Ich liczba zależy od tego, czy w podstawie choinki jest parzysta liczba gwiazdek, czy nieparzysta (liczba gwiazdek w podstawie to liczba podana przez użytkownika):

- jeżeli podstawa ma nieparzystą liczbę gwiazdek, to na szczycie zawsze będzie jedna gwiazdka,
- jeżeli podstawa ma parzystą liczbę gwiazdek, to na szczycie zawsze będą dwie gwiazdki.

Wiemy już zatem od wyświetlania ilu gwiazdek musimy zacząć.

A ile gwiazdek powinniśmy wyświetlić w każdym kolejnym rzędzie? Zauważ, że każdy kolejny rząd ma o 2 gwiazdki więcej od poprzedniego – będziemy więc zwiększać liczbę gwiazdek do wyświetlania w każdym kolejnym rzędzie o dwie.

A kiedy mamy przestać wyświetlać gwiazdki? Wtedy, gdy wyświetlimy tyle, ile miało być w podstawie choinki.

Wiemy już prawie wszystko – pozostaje jeszcze jedna kwestia: wcięcia na początku każdego rzędu, by gwiazdki na każdym poziomie były ładnie wyśrodkowane. Skąd mamy wiedzieć ile spacji powinniśmy wyświetlić na początku w każdym rzędzie?

Spacje, które należy wyświetlić w każdym rzędzie, można wyliczyć następującym wzorem (zaokrąglając wynik w dół):

```
(liczbaGwiazdekWPodstawie - liczbaGwiazdekWAktualnymRzedzie) / 2
```

Dzięki powyższym, możemy napisać rozwiązanie do tego zadania:

Pytania_i_zadania_rozwiazania/Rozdzial_08/Choinka.java

```
import java.util.Scanner;

public class Choinka {
  public static void main(String[] args) {
    System.out.print("Podaj liczbę gwiazdek w podstawie: ");
    int liczbaGwiazdekWPodstawie = pobierzLiczbe();

    int liczbaGwiazdekNaSzczycie =
        liczbaGwiazdekWPodstawie % 2 == 0 ? 2 : 1; // 1

    for ( // 2
        int gwiazdkiWRzedzie = liczbaGwiazdekNaSzczycie;
        gwiazdkiWRzedzie <= liczbaGwiazdekWPodstawie;
        gwiazdkiWRzedzie += 2
    ) {
      int liczbaSpacji = // 3
          (liczbaGwiazdekWPodstawie - gwiazdkiWRzedzie) / 2;
```

```
        // 4
        for (int i = 0; i < liczbaSpacji + gwiazdkiWRzedzie; i++) {
          System.out.print(i < liczbaSpacji ? " " : "*");
        }
        System.out.println(); // 5
      }
    }

    public static int pobierzLiczbe() {
      return new Scanner(System.in).nextInt();
    }
}
```

W linii `(1)` wyznaczamy liczbę gwiazdek na szczycie zgodnie z wcześniejszym opisem – jeżeli w podstawie ma być parzysta liczba gwiazdek, to na szczycie będą dwie gwiazdki. Jeżeli w podstawie będzie nieparzysta liczba gwiazdek, to na szczycie będzie jedna gwiazdka.

W linii `(2)` rozpoczyna się pętla wyświetlająca kolejne rzędy choinki. Ta pętla wypisuje w pierwszym obiegu tyle gwiazdek, ile jest na szczycie. Pętla będzie działać dopóki liczba gwiazdek do wypisania w rzędzie nie przekroczy liczby gwiazdek w podstawie. Na końcu każdego obiegu zwiększamy liczbę gwiazdek o `2`, ponieważ w każdym kolejnym rzędzie będą zawsze o dwie gwiazdki więcej, niż w poprzednim.

W każdym rzędzie przed gwiazdkami musimy wypisać odpowiednią liczbę spacji, aby gwiazdki były ładnie wyśrodkowane na ekranie. Liczba spacji do wypisania równa jest połowie różnicy gwiazdek w podstawie i w aktualnym rzędzie. Liczbę spacji do wypisania w aktualnym rzędzie wyznaczamy w linii `(3)`.

Pętla wewnętrzna, zaczynająca się w linii `(4)`, służy do wyświetlania zarówno znaków spacji, jak i gwiazdek. Łączna liczba znaków, które mamy wyświetlić, to suma liczby znaków spacji oraz gwiazdek do wypisania w aktualnym rzędzie. To, czy wyświetlić spację, czy gwiazdkę, uzależnione jest od wartości zmiennej `i` – spacje wypisujemy do czasu, aż wartość tej zmiennej nie przekroczy liczby spacji, które mieliśmy wyświetlić – gdy to się stanie, zaczniemy wypisywać gwiazdki.

Po zakończeniu wewnętrznej pętli wypisujemy na ekran znak nowej linii w linii `(5)`, by kolejny rząd gwiazdek został wypisany w następnej linii.

Dwa przykładowe uruchomienia tego programu:

```
Podaj liczbę gwiazdek w podstawie: 9
    *
   ***
  *****
 *******
*********
```

```
Podaj liczbę gwiazdek w podstawie: 20
         **
        ****
       ******
      ********
     **********
    ************
   **************
  ****************
 ******************
********************
```

TABLICE

9.1. Pytania

1. Do czego służą tablice?

Tablice służą do przechowywania ciągu elementów pewnego typu. Elementy te są przechowywane w ustalonej kolejności.

---

2. Jak definiuje się tablicę?

Tablice definiuje się tak, jak zwykłe zmienne, ale po nazwie typu należy dodać nawiasy kwadratowe. Dozwolone jest także umieszczenie nawiasów kwadratowych nie po nazwie typu, lecz po nazwie zmiennej:

```
int[] tablica;
int tablica[];
```

---

3. Jak utworzyć tablicę?

Mamy kilka opcji aby utworzyć tablicę – możemy:

- skorzystać ze słowa kluczowego `new`, podając ile elementów tablica będzie mogła przechowywać – elementy takiej tablicy w momencie jej tworzenia są wypełniane domyślnymi wartościami typu, którego elementy ta tablica będzie zawierać,
- użyć inicjalizatora tablicowego w momencie definicji zmiennej-tablicy – tablica zostanie wypełniona wartościami zawartymi w nawiasach klamrowych; w tym przypadku nie korzystamy ze słowa kluczowego `new`,
- skorzystać z rozwiązania będącego połączeniem dwóch powyższych – utworzenia nowej tablicy za pomocą słowa kluczowego `new` wraz z podaniem elementów, z których tablica ma się składać.

```
// pierwszy sposób
int[] liczbyCalkowite = new int[5];

// drugi sposób
String[] slowa = { "Ala", "ma", "kota" };

// trzeci sposób
String[] papierKamienNozyce;
papierKamienNozyce =
    new String[] { "papier", "kamień", "nożyce" };
```

4. Czy rozmiar tablicy można zmienić?

Nie, rozmiaru raz utworzonej tablicy nie można zmienić.

---

5. Jak odnieść się do danego elementu tablicy?

Aby odnieść się do elementu tablicy, korzystamy z nawiasów kwadratowych po nazwie zmiennej tablicowej. W nawiasach kwadratowych należy umieścić indeks elementu, do którego chcemy się odnieść. Indeksy elementów tablic w języku Java zaczynają się od `0`, a nie `1`. Przykład wypisania drugiego element tablicy:

```
System.out.println(mojaTablica[1]);
```

---

6. Jak odnieść się do pierwszego, a jak do ostatniego elementu tablicy?

Pierwszy element tablicy ma zawsze indeks `0`.

Ostatni element tablicy ma indeks równy `długość_tablicy - 1`. Możemy więc skorzystać z pola `length` tablicy, aby wyliczyć indeks ostatniego elementu:

```
System.out.println(mojaTablica[0]); // pierwszy element
System.out.println(
    mojaTablica[mojaTablica.length - 1] // ostatni element
);
```

---

7. Czy poniższy kod jest poprawny?

```
int[] tablica = { 1, 2, 3 };

System.out.println(tablica[3]);
```

Jest to poprawnie zapisany kod w języku Java, tzn. nie ma w nim błędów składniowych, więc program z tym fragmentem kodu skompiluje się bez błędów. Z drugiej strony, ten kod zawiera błąd, ponieważ próbujemy odnieść się do czwartego elementu tablicy, która ma tylko trzy elementy. Ten fragment kodu spowoduje błąd działania programu: `ArrayIndexOutOfBoundsException`.

8. Jak sprawdzić ile elementów znajduje się w tablicy?

Aby dowiedzieć się, z ilu elementów składa się tablica, należy odczytać wartość pola `length`, które ma każda tablica:

```
int[] liczby = { 1, 5, -20 };
System.out.println(liczby.length); // wypisze 3
```

---

9. Jaki będzie wynik działania poniższego fragmentu kodu gdy wartość zmiennej `szukanaLiczba` będzie równa `0`, a jaki, gdy będzie miała wartość `500`?

```
boolean znaleziona = false;
int[] tablica = { -20, 105, 0, 26, -99, 7, 1026 };

int szukanaLiczba = ?; // pewna wartość

for (int i = 0; i <= tablica.length; i++) {
  if (tablica[i] == szukanaLiczba) {
    znaleziona = true;
    break; // znaleźlismy liczbę - możemy przerwać pętlę
  }
}

if (znaleziona) {
  System.out.println(
      "Liczba " + szukanaLiczba + " została znaleziona."
  );
} else {
  System.out.println(
      "Liczba " + szukanaLiczba + " nie została znaleziona."
  );
}
```

W przypadku, gdy będziemy szukać liczby `0`, na ekranie zobaczymy komunikat `"Liczba 0 została znaleziona."`. Gdy będziemy szukać liczby `500`, to na ekranie zobaczymy komunikat:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 7
```

Wykonanie programu zakończy się błędem – wynika to z niepoprawnego warunku pętli. Liczby `500` nie ma w tablicy, więc pętla przejdzie przez wszystkie elementy. W warunku pętli powinniśmy użyć operatora < zamiast <=. Użycie operatora <= powoduje, że w ostatnim obiegu pętli wychodzimy poza zakres tablicy.

10. Czy poniższy kod się skompiluje? Jeżeli tak, to czy wykona się bez błędów?

```
int[][] tablica2d = new int[3][5];

tablica2d[3][1] = 1;
```

Ten kod skompiluje się bez błędów, jednak jego wykonanie zakończy się błędem `ArrayIndexOutOfBoundsException`, ponieważ tablica `tablica2d` ma tylko trzy elementy w „pierwszym wymiarze", a w powyższym fragmencie kodu próbujemy odnieść się do czwartego elementu pierwszego wymiaru tej tablicy.

---

11. Czy poniższy kod jest poprawny?

```
String[] powitanie = { "Witaj", "Świecie" };

for (int i = 0; i < powitanie.length(); i++) {
  System.out.println(powitanie[i] + " ");
}
```

Ten kod się nie skompiluje, ponieważ odnosząc się do rozmiaru tablicy nie powinniśmy dodawać nawiasów `()` po polu `length`.

---

12. Jaki będzie wynik działania poniższego fragmentu kodu?

```
String[] slowa = new String[5];
System.out.println("Pierwsze słowo: " + slowa[0]);
```

Elementy tablic są wypełniane domyślnymi wartości danego typu. W przypadku typów złożonych, do których należy typ `String`, domyślną wartością jest specjalna wartość `null`, więc na ekranie zobaczymy następujący komunikat:

```
Pierwsze słowo: null
```

---

13. Jaki będzie wynik działania poniższego fragmentu kodu?

```
double[] a = { 3.14, 2.44, 0.1 };
double[] b = { 3.14, 2.44, 0.1 };

if (a == b) {
  System.out.println("Tablice są takie same.");
} else {
  System.out.println("Tablice nie są takie same.");
}
```

Na ekranie zobaczymy komunikat „Tablice nie są takie same.”, ponieważ tablice powinniśmy porównywać poprzez sprawdzenie elementów na tych samych pozycjach, a nie poprzez użycie operatora ==. Użycie tego operatora odpowiada na pytanie: „Czy dwie zmienne-tablice wskazują na dokładnie tę samą tablicę w pamięci?”, a nie na pytanie „Czy dwie zmienne-tablice wskazują na tablice, które mają taki sam rozmiar i takie same elementy w tej samej kolejności”.

---

14. Jak zmodyfikować w linii `(2)` tablicę przypisaną do zmiennej `znaki` w linii `(1)`?

```
char[] znaki = { 'k', 'o', 't' }; // 1
znaki = null;
// 2
```

W linii `(2)` nie można już zmienić zawartości tablicy, na którą w linii `(1)` pokazywała zmienna `znaki`. Po przypisaniu wartości `null` do zmiennej `znaki` nic już nie wskazuje na tablicę utworzoną w linii `(1)`, więc nie mamy jak się do niej odnieść.

---

15. Która z poniższych tablic jest zdefiniowana/utworzona niepoprawnie i dlaczego?

```
// błąd: brakuje [] po int
int liczby = { 1, 2, 3 };

// błąd: stringi zapisujemy w "", a nie w ''
String[] litery = { 'a', 'b', 'c' };

// błąd: brakuje rozmiaru tablicy w []
String[] slowa = new String[];
// błąd: inicjalizacja za pomocą { } może być użyta jedynie
// podczas definicji tablicy
slowa = { "Ala", "ma", "kota" };
```

```
// poprawna tablica
double[] rzeczywiste = new double[] { 3.14, 2.44, 20 };

// błąd: niespójny typ double <-> int
double[] innaTablica = new int[3];

// błąd: podczas tworzenia tablicy za pomoca new
// i podawania elementow w {} nie należy umieszczać
// w nawiasach [] rozmiaru tablicy
int[] tablica = new int[5] { 1, 10, 100 };

// poprawna tablica
double[] kolejnaTablica = new double[3];
// błąd: w ten sposób po utworzeniu tablicy nie można
// jej przypisać wartości
kolejnaTablica = { 5, 10, 15 };

// poprawna tablica z tylko jednym elementem
String[] tab = { "Ala ma kota" };
```

16. Do czego służy pętla `for`-each i jak się z niej korzysta?

Pętla `for`-each służy do iterowania po tablicach (i innych kolekcjach). Jest to krótka i wygodna alternatywa do przechodzenia po elementach tablicy za pomocą pętli `for`.

Składnia tej pętli jest następująca:

```
for (typ nazwa_zmienna : zmienna_tablica) {
  // instrukcje
}
```

W pętli `for`-each definiujemy zmienną, która w każdym obiegu będzie miała przypisany kolejny element z podanej tablicy.

---

17. Co zostanie wypisane w ramach działania poniższego fragmentu kodu?

```
int[] liczby = { 0, 1, 2, -1 };

for (int i : liczby) {
  System.out.println(liczby[i] + ", ");
}
```

Jest to podchwytliwe pytanie – pętla `for`-each jest w nim niepoprawnie użyta. Zamiast po prostu wypisać wartość zmiennej pętli `i`, która w każdym obiegu przyjmuje wartość kolejnego elementu tablicy `liczby`, korzystamy z tej wartości jako indeksu

elementów tablicy `liczby`. W związku z tym, że akurat pierwsze wartości przechowywane w tablicy `liczby` to `0`, `1`, `2`, to odniesiemy się do, kolejno, pierwszego, drugiego, oraz trzeciego elementu tej tablicy i na ekranie zobaczymy liczby `0`, `1`, `2`.

Jednakże, w ostatnim obiegu pętli, zmienna `i` będzie miała wartość `-1` – jest to nieprawidłowy indeks tablicy, co spowoduje wystąpienie błędu `ArrayIndexOutOfBoundsException`. Finalnie, na ekranie zobaczymy:

```
0,
1,
2,
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index -1 out of bounds for length 4
```

Powyższy kod poprawnie powinien zostać zapisany w następujący sposób (wtedy na ekranie zobaczylibyśmy `0`, `1`, `2`, `-1`):

```
int[] liczby = { 0, 1, 2, -1 };

for (int i : liczby) {
  System.out.println(i + ", ");
}
```

---

18. Co wypisze na ekran poniższy fragment kodu?

```
int[] liczby = new int[3];

System.out.println(liczby[1]);
```

Na ekranie zobaczymy liczbę `0`. Co prawda nie ustawiliśmy w tablicy liczby żadnych wartości, ale elementy tablic są inicjalizowane domyślnymi wartościami typu, którego wartości tablica może przechowywać. W przypadku typu `int` jest to `0`.

---

19. Czy poniższy fragment kodu jest poprawny?

```
int[][] tablicaDwuwymiarowa =
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 };
```

Nie, ta tablica nie jest zainicjalizowana w poprawny sposób – brakuje jeszcze jednej pary nawiasów klamrowych, ponieważ jest to tablica dwuwymiarowa. Poprawna inicjalizacja:

```
int[][] tablicaDwuwymiarowa = {
    { 1, 2, 3 },
    { 4, 5, 6 },
    { 7, 8, 9 }
};
```

---

20. Jaki będzie wynik działania poniższego fragmentu kodu?

```
String[] slowa = new String[5];
System.out.println(
    "Długość pierwszego elementu: " + slowa[0].length()
);
```

Ten fragment kodu spowoduje zakończenie się działania programu błędem `NullPointerException`. Nie zainicjalizowaliśmy elementów tablicy `slowa` żadnymi wartościami typu `String`, więc domyślnie wszystkie elementy mają wartość `null`. Próba odwołania się do metody na wartości typu `null` generuje wspomniany błąd.

---

21. Ile tablic zostało utworzonych w poniższym fragmencie kodu?

```
int[] liczby = { 10, 20, 30 };
int[] tezLiczby = liczby;
int[] inneLiczby = new int[10];
```

W tym fragmencie kodu utworzone zostały dwie tablice – pierwszą przypisaliśmy do zmiennej `liczby`, a drugą – do zmiennej `inneLiczby`. Zmienna `tezLiczby` wskazuje na tę samą tablicą, na którą wskazuje zmienna `liczby`, więc w tym fragmencie kodu mamy trzy zmienne-tablice i dwie faktycznie utworzone tablice.

---

22. Co zostanie wypisane przez poniższy fragment kodu?

```
String[] slowa = { "Ala", "ma", null };
String[] inneSlowa = { "Ania", "ma", null };

if (slowa[2] == inneSlowa[2]) {
  System.out.println("Takie same.");
} else {
  System.out.println("Inne.");
}
```

Na ekranie zobaczymy komunikat „Takie same.”. W tym fragmencie kodu porównujemy do siebie trzeci element tablicy `slowa` i trzeci element tablicy `inneSlowa`. Oba te elementy to wartości `null`. W języku Java dwa nulle traktowane są jako takie same wartości, więc porównanie ich do siebie za pomocą operatora == daje w wyniku `true`.

9.2. Zadania

9.2.1. Co druga wartość tablicy

Napisz program, który wypisze co drugi element zdefiniowanych przez Ciebie tablic. Pierwsza tablica powinna mieć parzystą liczbę elementów, a druga nieparzystą.

Nie ma różnicy, czy wypisujemy co drugi element z tablicy o parzystej czy nieparzystej liczbie elementów. Musimy tylko odpowiednio zapisać instrukcję kroku pętli `for` – tak, aby zmienna pętli zwiększała się o `2`, a nie o `1`. Przykładowe rozwiązanie tego zadania:

Pytania_i_zadania_rozwiazania/Rozdzial_09/CoDrugaWartoscTablicy.java

```
public class CoDrugaWartoscTablicy {
  public static void main(String[] args) {
    int[] parzysta = { 1, 10, 15, 0, 100, 20 };
    int[] nieparzysta = { 5, 7, 9 };

    for (int i = 0; i < parzysta.length; i += 2) {
      System.out.print(parzysta[i] + ", ");
    }

    System.out.println();

    for (int i = 0; i < nieparzysta.length; i += 2) {
      System.out.print(nieparzysta[i] + ", ");
    }
  }
}
```

Wynik uruchomienia tego programu jest następujący:

```
1, 15, 100,
5, 9,
```

9.2.2. Największa liczba w tablicy

Napisz program, który wypisze największą liczbę z tablicy. Tablicę zainicjalizuj przykładowymi wartościami.

Aby znaleźć największą liczbę w tablicy, musimy porównać jej kolejne elementy do poprzedniego, największego elementu. Jeżeli kolejna sprawdzana liczba będzie większa od poprzednio znalezionej największej liczby, to zapiszemy tę aktualną wartość jako największą.

Trzeba tylko rozwiązać jeden problem – jaką wartość powinniśmy użyć jako pierwszą „największą" liczbę, do której na początku będziemy porównywać wartości z tablicy? Możemy jako tę liczbę wybrać pierwszy element tablicy. Następnie, w pętli przejdziemy przez pozostałe elementy i będziemy je porównywać z zapisaną, największą liczbą. Jeżeli aktualnie sprawdzana wartość z tablicy będzie większa, niż poprzednio zapamiętana największa liczba, to zapiszemy ten aktualny element jako największy itd., aż przejdziemy przez całą tablicę:

Pytania_i_zadania_rozwiazania/Rozdzial_09/NajwiekszaWartoscWTablicy.java

```
public class NajwiekszaWartoscWTablicy {
  public static void main(String[] args) {
    int[] liczby = { 1, -20, 100, 40, -15 };

    int najwiekszaLiczba = liczby[0]; // 1

    for (int i = 1; i < liczby.length; i++) {
      if (liczby[i] > najwiekszaLiczba) { // 2
        najwiekszaLiczba = liczby[i];
      }
    }

    System.out.println(
        "Największa liczba to: " + najwiekszaLiczba
    );
  }
}
```

Na początku `(1)` zakładamy, że największą liczbą jest wartość pierwszego elementu tablicy `liczby`. Dzięki temu w pętli w linii `(2)` mam już jakąś wartość, do której możemy porównywać elementy tablicy. Zmienną pętli `i` inicjalizujemy wartością `1`, ponieważ chcemy pominąć pierwszy element – został on użyty do nadania wartości zmiennej `najwiekszaLiczba` w linii `(1)`, więc może zostać pominięty.

Takie rozwiązanie ma jedną wadę – zakładamy, że tablica nie jest pusta. Ten przypadek jest problematyczny, bo jeżeli naszym zadaniem jest znalezienie największej liczby w tablicy, która nie ma elementów, to nie możemy wykonać tego zadania. W przyszłości nauczysz się korzystać z typu `Optional`, którego wartości mogą zawierać w sobie pewną inną wartość, ale nie muszą. Ten typ przydaje się w właśnie w takich sytuacjach, jak szukanie największe elementu, gdy tablica jest pusta – możemy wtedy zwrócić „pustą" wartość typu `Optional`, dając znać użytkownikowi naszego kodu, że nie udało się wykonać danej operacji. Ta wartość będzie przydatna

podczas pisania i używania metod, które są tematem ostatniego rozdziału pierwszego tomu *„Javy od podstaw”*, jednakże typ `Optional` będzie opisany dopiero w jednym z kolejnych tomów.

Przykładowe uruchomienie tego programu:

```
Największa liczba to: 100
```

9.2.3. Słowa z tablicy wielkimi literami

Napisz program, w którym zdefiniujesz tablicę wartości typu `String` i zainicjalizujesz ją przykładowymi wartościami. Skorzystaj z pętli `for`-each aby wypisać wszystkie wartości tablicy ze wszystkimi literami zamienionymi na wielkie. Skorzystaj z metody `toUpperCase` wartości typu `String`. Wartości powinny być wypisane w jednej linii i być od siebie oddzielone pojedynczym znakiem spacji.

Rozwiązanie tego zadania może wyglądać następująco:

Pytania_i_zadania_rozwiazania/Rozdzial_09/SlowaZTablicyWielkimiLiterami.java

```
public class SlowaZTablicyWielkimiLiterami {
  public static void main(String[] args) {
    String[] slowa = { "Ala", "ma", "kota", "i", "psa" };

    for (String slowo : slowa) {
      System.out.print(slowo.toUpperCase() + " ");
    }
  }
}
```

Przykładowe uruchomienie tego programu:

```
ALA MA KOTA I PSA
```

9.2.4. Silnia liczb w tablicy

Napisz program, który pobierze od użytkownika pięć liczb naturalnych, zapisze je w tablicy, a następnie policzy i wypisze silnię każdej z pobranych liczb. Każda liczba powinna być większa bądź równa zero – jeżeli któraś z liczb będzie nieprawidłowa, użytkownik powinien podać ją ponownie. Silnia liczby N to iloczyn liczb od `1` do N. Dla przykładu, silnia liczby `5` to: `1 * 2 * 3 * 4 * 5 = 120`. Silnia liczby `0` to `1`.

W proponowanym rozwiązaniu najpierw pobieramy od użytkownika pięć liczb. W pętli `while` wypisujemy komunikat z prośbą o podanie wartości i zapisujemy ją w tablicy `liczby` pod aktualnym indeksem, który wyznacza zmienna `indeks`. Jeżeli użytkownik poda nieprawidłową liczbę naturalną, to wypiszemy stosowny komunikat (linia `(1)`). W takim przypadku nie zwiększymy wartości zmiennej `indeks`, którą używamy w warunku pętli. Dzięki temu, gdy użytkownik poda nieprawidłową liczbę, zostanie o nią poproszony ponownie.

Pytania_i_zadania_rozwiazania/Rozdzial_09/SilniaLiczbWTablicy.java

```
import java.util.Scanner;

public class SilniaLiczbWTablicy {
  public static void main(String[] args) {
    int[] liczby = new int[5];

    System.out.println("Podaj pięć liczb naturalnych.");

    int indeks = 0;
    while (indeks < liczby.length) {
      System.out.printf("Liczba #%d: ", indeks + 1);
      liczby[indeks] = pobierzLiczbe();

      if (liczby[indeks] < 0) {
        System.out.println( // 1
            "Nieprawidłowa liczba naturalna."
        );
      } else {
        indeks++;
      }
    }

    for (int x : liczby) { // 2
      int silnia = 1;

      for (int i = 1; i <= x; i++) {
        silnia *= i;
      }

      System.out.printf(
          "Silnia liczby %d wynosi %d.\n", x, silnia
      );
    }
  }

  public static int pobierzLiczbe() {
```

```
        return new Scanner(System.in).nextInt();
    }
}
```

W dalszej części programu przechodzimy przez wszystkie pobrane liczby w pętli `for`-each, która zaczyna się w linii `(2)`. W pętli wewnętrznej liczymy silnię dla aktualnie przetwarzanej liczby, po czym wypisujemy ją na ekran. Przykładowe uruchomienie tego programu:

```
Podaj pięć liczb naturalnych.
Liczba #1: -1
Nieprawidłowa liczba naturalna.
Liczba #1: 1
Liczba #2: 2
Liczba #3: 3
Liczba #4: 4
Liczba #5: -5
Nieprawidłowa liczba naturalna.
Liczba #5: 10
Silnia liczby 1 wynosi 1.
Silnia liczby 2 wynosi 2.
Silnia liczby 3 wynosi 6.
Silnia liczby 4 wynosi 24.
Silnia liczby 10 wynosi 3628800.
```

9.2.5. Suma liczb parzystych i nieparzystych

Napisz program, w którym zdefiniujesz tablicę liczb całkowitych. Napisz kod, który dla tej tablicy odpowie na pytanie: czy suma liczb parzystych, zawartych w tej tablicy, jest większa, równa, bądź mniejsza, niż suma liczb nieparzystych z tej tablicy. Wypisz obie sumy na ekran.

Dla przykładu, dla tablicy złożonej z elementów:

```
99, 10, -20, 100, 11, 1, 8
```

odpowiedzią jest:

```
Suma liczb parzystych wynosi: 98
Suma liczb nieparzystych wynosi: 111
Suma liczb parzystych w tablicy jest mniejsza niż suma liczb nieparzystych.
```

Na początku metody `main` definiujemy przykładową tablicę oraz dwie zmienne, do których będziemy dodawać odpowiednie liczby.

W pętli zaczynającej się w linii `(1)` w zależności od tego, czy aktualnie przetwarzana liczba jest parzysta, czy nie, dodamy jej wartość do zmiennej `sumaParzystych` lub do zmiennej `sumaNieparzystych`. Do sprawdzenia, czy liczba jest parzysta, posłuży nam operator reszty z dzielenia (linia `(2)`).

Pytania_i_zadania_rozwiazania/Rozdzial_09/SumaLiczbParzystychINieparzystych.java

```
public class SumaLiczbParzystychINieparzystych {
  public static void main(String[] args) {
    int[] liczby = { 99, 10, -20, 100, 11, 1, 8 };

    int sumaParzystych = 0;
    int sumaNieparzystych = 0;

    for (int liczba : liczby) { // 1
      if (liczba % 2 == 0) { // 2
        sumaParzystych += liczba;
      } else {
        sumaNieparzystych += liczba;
      }
    }

    System.out.println(
        "Suma liczb parzystych wynosi: " + sumaParzystych
    );
    System.out.println(
        "Suma liczb nieparzystych wynosi: " +
        sumaNieparzystych
    );

    if (sumaParzystych > sumaNieparzystych) {
      System.out.println("""
          Suma liczb parzystych w tablicy \
          jest większa niż suma liczb nieparzystych.""
      );
    } else if (sumaParzystych == sumaNieparzystych){
      System.out.println("""
          Suma liczb parzystych w tablicy \
          jest równa sumie liczb nieparzystych."""
      );
    } else {
      System.out.println("""
          Suma liczb parzystych w tablicy \
          jest mniejsza niż suma liczb nieparzystych."""
```

```
        );
      }
    }
}
```

Po zakończeniu pętli wyświetlamy sumę liczb parzystych i nieparzystych, a także komunikat informujący, czy suma liczba parzystych jest większa, równa, czy mniejsza od sumy liczb nieparzystych:

```
Suma liczb parzystych wynosi: 98
Suma liczb nieparzystych wynosi: 111
Suma liczb parzystych w tablicy jest mniejsza niż suma liczb nieparzystych.
```

9.2.6. Wypisywanie dwuwymiarowej tablicy

Napisz program, w którym zdefiniujesz dwuwymiarową tablicę wartości typu `int`. Wypisz tę tablicę na ekran traktując każdą „zagnieżdżoną” tablicę jako kolejny rząd wartości do wypisana na ekran. Dla każdego parzystego rzędu wartości w nim zawarte powinny zostać wypisane od końca.

Przykład – tablica:

```
int[][] tab2d = {
    { 10, 20, 30, 40, 50 },
    { 1, 2, 3, 4, 5 },
    { 5, 10, 15, 20, 25 },
    { 100, 200, 300, 400, 500 },
    { -1, -2, -3, -4, -5 }
};
```

powinna zostać wypisana na ekran w następujący sposób:

```
10 20 30 40 50
5 4 3 2 1
5 10 15 20 25
500 400 300 200 100
-1 -2 -3 -4 -5
```

Na początku metody `main` zdefiniowana jest przykładowy tablica – taka sama, jak w treści zadania.

W pętli zaczynającej się w linii `(1)` przechodzimy przez pierwszy wymiar naszej dwuwymiarowej tablicy. Jeżeli numer wyświetlanego rzędu jest parzysty, co

sprawdzamy w linii (2), to zgodnie z wymaganiami zadania zawartość tego „rzędu" zostanie wypisana od końca. Posłuży nam w tym celu wewnętrzna pętla z linii (3) z odpowiednio zapisanymi: instrukcją inicjalizującą, warunkiem, oraz instrukcją kroku.

Pytania_i_zadania_rozwiazania/Rozdzial_09/WypisywanieDwuwymiarowejTablicy.java

```
public class WypisywanieDwuwymiarowejTablicy {
  public static void main(String[] args) {
    int[][] tab2d = {
        { 10, 20, 30, 40, 50 },
        { 1, 2, 3, 4, 5 },
        { 5, 10, 15, 20, 25 },
        { 100, 200, 300, 400, 500 },
        { -1, -2, -3, -4, -5 }
    };

    for (int rzad = 0; rzad < tab2d.length; rzad++) { // 1
      if ((rzad + 1) % 2 == 0) { // 2
        // 3
        for (int j = tab2d[rzad].length - 1; j >= 0; j--) {
          System.out.print(tab2d[rzad][j] + " ");
        }
      } else {
        for (int liczba : tab2d[rzad]) { // 4
          System.out.print(liczba + " ");
        }
      }
      System.out.println(); // 5
    }
  }
}
```

Jeżeli aktualny rząd nie jest parzysty, to w linii (4) skorzystamy z prostej pętli `for`-each w celu wypisania na ekran elementów „rzędu" od początku do końca. Po zakończeniu pętli wewnętrznych wypisujemy w linii (5) znak nowej linii, aby kolejny „rząd" wartości wypisywany był od nowej linii, jak widać poniżej:

```
10 20 30 40 50
5 4 3 2 1
5 10 15 20 25
500 400 300 200 100
-1 -2 -3 -4 -5
```

9.2.7. Podział tablicy

Napisz program, w którym utworzysz jedną tablicę o nazwie `tablicaZrodlowa`. Utwórz dwie nowe tablice:

- tablica `pierwszaPolowa` powinna zawierać elementy z pierwszej połowy tablicy `tablicaZrodlowa`,
- tablica `drugaPolowa` powinna zawierać elementy z drugiej połowy tablicy `tablicaZrodlowa`.

Jeżeli `tablicaZrodlowa` będzie miała nieparzystą liczbę elementów, to `pierwszaTablica` powinna zawierać o jeden element więcej.

Przykład: gdy `tablicaZrodlowa` zawiera elementy `1`, `20`, `3`, `40`, to do tablicy `pierwszaPolowa` powinny zostać wpisane wartości `1`, `20`, a do tablicy `drugaPolowa`: `3`, `40`.

Przykład: gdy elementami `tablicaZrodlowa` są `1`, `20`, `100`, `3`, `40`, to do tablicy `pierwszaPolowa` powinny zostać wpisane wartości `1`, `20`, `100`, a do `drugaPolowa`: `3`, `40`.

To zadanie możemy rozwiązać albo sami przepisując elementy z tablicy źródłowej do każdej z jej „tablic-połówek", albo skorzystać z metody `Arrays.copyOfRange`, którą poznałeś(-aś) w rozdziale o tablicach. Poniżej znajdziesz dwie wersje rozwiązania – pierwsza skorzysta z pętli, a druga ze wspomnianej metody.

Zgodnie z wymaganiami zadania, gdy tablica źródłowa ma nieparzystą liczbę elementów, to po podzieleniu tej tablicy pierwsza część powinna zawierać o jeden element więcej. Musimy więc w pierwszej kolejności określić ile elementów będzie miała pierwsza pod-tablica.

Na początku metody `main`, w linii `(1)`, zakładamy, że tablica źródłowa jest parzysta – wtedy pierwsza pod-tablica i druga pod-tablica będą miał po połowie elementów tablicy źródłowej. Jednakże, w linii `(2)` sprawdzamy, czy liczba elementów tablicy źródłowej jest nieparzysta – w tym przypadku pierwsza pod-tablica będzie o jeden element większa, niż druga, więc zwiększymy wartość zmiennej `dlugoscPierwszejPolowy` o jeden.

Znając długość pierwszej pod-tablicy możemy wyznaczyć rozmiar drugiej – tę długość obliczamy i zapisujemy w linii `(3)` w zmiennej `dlugoscDrugiejPolowy`.

Znając już długości każdej z pod-tablic, możemy je utworzyć (linia `(4)`).

Pytania_i_zadania_rozwiazania/Rozdzial_09/PodzialTablicy.java

```
import java.util.Arrays;

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

```
    int[] tablicaZrodlowa = { 1, 20, 3, 40 };

    int dlugoscPierwszejPolowy =
        tablicaZrodlowa.length / 2;

    if (tablicaZrodlowa.length % 2 == 1) { // 2
      dlugoscPierwszejPolowy++;
    }

    int dlugoscDrugiejPolowy = // 3
        tablicaZrodlowa.length - dlugoscPierwszejPolowy;

    // 4
    int[] pierwszaPolowa = new int[dlugoscPierwszejPolowy];
    int[] drugaPolowa = new int[dlugoscDrugiejPolowy];

    for (int i = 0; i < tablicaZrodlowa.length; i++) {
      if (i < dlugoscPierwszejPolowy) { // 5
        pierwszaPolowa[i] = tablicaZrodlowa[i];
      } else { // 6
        drugaPolowa[i - dlugoscPierwszejPolowy] =
            tablicaZrodlowa[i];
      }
    }

    System.out.println(
        "Pierwsza połowa: " + Arrays.toString(pierwszaPolowa)
    );
    System.out.println(
        "Druga połowa: " + Arrays.toString(drugaPolowa)
    );
  }
}
```

Pozostaje nam jeszcze przepisać elementy z tablicy źródłowej do pod-tablic. To, do której pod-tablicy mamy wpisać aktualny element, zależne jest od długości pierwszej tablic i wartości zmiennej pętli `i`. Gdy wartość zmiennej `i` jest mniejsza, niż długość pierwszej pod-tablicy, to przepisujemy wartość do pierwszej pod-tablicy (linia `(5)`). Gdy zmienna `i` przekroczy zakres indeksów pierwszej pod-tablicy, to zaczniemy przepisywanie elementów do drugiej pod-tablicy (sekcja `else` z linii `(6)`).

Istotne jest tutaj to, że odnosząc się do elementu drugiej pod-tablicy w sekcji `else` musimy w odpowiedni sposób obliczyć indeks elementu, do którego wartość z tablicy źródłowej ma być wpisana. Zauważ, że zaczynamy przepisywać wartości do drugiej pod-tablicy, gdy wartość `i` będzie równa wartości `dlugoscPierwszejTablicy` (bo wtedy warunek z linii `(5)` przestanie być spełniony i przejdziemy do sekcji `else`).

Oznacza to, że w tym momencie powinniśmy wpisać wartość z tablicy źródłowej do pierwszego elementu drugiej pod-tablicy – jego indeks to `0`. Aby obliczyć aktualny indeks w drugiej pod-tablicy wystarczy, że od wartości zmiennej `i` odejmiemy długość pierwszej tablicy:

```
drugaPolowa[i - dlugoscPierwszejPolowy] =
    tablicaZrodlowa[i];
```

W ten sposób przechodząc przez drugą połowę źródłowej tablicy będzie jej wartościami wypełniać drugą pod-tablicę zaczynając od samego początku tej drugiej pod-tablicy. Na końcu wypisujemy na ekran zawartość obu pod-tablic:

```
Pierwsza połowa: [1, 20, 100]
Druga połowa: [3, 40]
```

Drugie rozwiązanie tego zadania korzysta z metody `Arrays.copyOfRange`, która tworzy kopię tablicy mającą określony zakres elementów z oryginału. W tym rozwiązaniu dodatkowo wyznaczyłem długość pierwszej połowy w inny sposób – skorzystałem z metody `Math.ceil`, aby zaokrąglić wynik dzielenia długości tablicy źródłowej przez dwa do góry – dzięki temu nie jest potrzebna dodatkowa instrukcja warunkowa, która sprawdza, czy liczba elementów jest parzysta, czy nieparzysta. Dodatkowo, zmienna `dlugoscDrugiejPolowy` nie jest już potrzebna, bo rozmiar drugiej pod-tablicy wyznaczy dla nas metoda `copyOfRange` dzięki odpowiednio przekazanym jej argumentom.

Pytania_i_zadania_rozwiazania/Rozdzial_09/PodzialTablicyCopyOfRange.java

```
import java.util.Arrays;

public class PodzialTablicyCopyOfRange {
  public static void main(String[] args) {
    int[] tablicaZrodlowa = { 1, 20, 100, 3, 40 };

    int dlugoscPierwszejPolowy =
        (int) Math.ceil(tablicaZrodlowa.length / 2.0);

    int[] pierwszaPolowa = Arrays.copyOfRange( // 1
        tablicaZrodlowa, 0, dlugoscPierwszejPolowy
    );

    int[] drugaPolowa = Arrays.copyOfRange( // 2
        tablicaZrodlowa,
        dlugoscPierwszejPolowy,
        tablicaZrodlowa.length
    );
```

```
        System.out.println(
            "Pierwsza połowa: " + Arrays.toString(pierwszaPolowa)
        );
        System.out.println(
            "Druga połowa: " + Arrays.toString(drugaPolowa)
        );
    }
}
```

Jak widzisz, ta wersja jest dużo krótsza – przepisywanie elementów oddelegowane zostało do metody `copyOfRange`, która użyta jest w liniach `(1)` i `(2)`.

W linii `(1)` tworzymy kopię zawierającą tyle elementów od początku tablicy `tablicaZrodlowa`, ile wyznaczyliśmy, że ma być w pierwszej pod-tablicy.

W linii `(2)` tworzymy kopię, która będzie zawierała elementy zaczynając od indeksu, który wyznacza długość pierwszej pod-tablicy, aż do końca tablicy źródłowej.

Na końcu wypisujemy na ekran zawartość obu pod-tablic:

```
Pierwsza połowa: [1, 20, 100]
Druga połowa: [3, 40]
```

9.2.8. Czy słowa składają się z takich samych liter?

Napisz program, który pobierze od użytkownika dwa słowa. Program powinien sprawdzić, czy te słowa składają się z takich samych liter. Liczba liter nie musi się zgadzać, tzn. uznajemy, że słowa `"mama"` i `"ma"` składają się z takich samych liter (tylko liter `'m'` i `'a'`). Wielkie i małe litery powinny być uznane za takie same litery.

Typ `String` posiada metodę `toCharArray`, która zwraca tablicę znaków, z których ten `String` się składa. Przyda Ci się ona w rozwiązaniu tego zadania.

Zanim przejdziemy do omówienia kodu źródłowego zastanówmy się w jaki sposób możemy rozwiązać to zadanie. Zgodnie z opisanym wymaganiem wielkość liter ma nie mieć znaczenia, więc będziemy musieli skorzystać np. z metody `toLowerCase`, aby zamienić litery w obu pobranych słowach na małe litery.

W opisie zadania wspomniana jest metoda `toCharArray`. Ta metoda, wywołana na pewnym stringu, zwraca tablicę, której elementami są kolejne znaki w tym stringu. Dla przykładu, mając łańcuch tekstowy `"mama"`, wynikiem poniższej instrukcji:

```
"mama".toCharArray()
```

jest tablica z czterema elementami typu `char` w następującej kolejności:

```
{ 'm', 'a', 'm', 'a' }
```

Jeżeli pobierzemy znaki obu słów, to możemy sprawdzić, czy każdy znak znajdujący się w tablicy znaków pierwszego słowa jest zawarty w drugiej tablicy i na odwrót. Takie rozwiązanie dałoby odpowiedź czy te słowa składają się z tych samych znaków, jednak dla dużych zestawów danych byłoby ono niewydajne.

Możemy wykonać to zadanie w inny sposób – najpierw posortujemy obie tablice ze znakami, następnie usuniemy z nich duplikaty, a końcu porównamy do siebie te tablice – jeżeli pozostaną w nich te same znaki, będzie to oznaczało, że oba słowa składają się dokładnie z takiego samego zestawu znaków. Ten algorytm będzie działał szybciej dla dużych zestawów danych niż poprzednio opisany.

Do posortowania tablic możemy użyć poznanej w rozdziale o tablicach metody `sort` z klasy `Arrays` z Biblioteki Standardowej Java. Do porównania tablic użyjemy metody `equals`, która także wchodzi w skład tej klasy. Pozostaje nam jedynie usunąć zduplikowane znaki z obu tablic.

Aby usunąć duplikaty z posortowanej tablicy możemy wykonać następujące kroki:

- utworzyć tymczasową tablicę o rozmiarze oryginalnej tablicy,
- przejść w pętli przez elementy tablicy źródłowej i przepisać je do tymczasowej tablicy z pominięciem duplikatów, zliczając przy okazji ile unikalnych znaków znaleźliśmy,
- utworzyć nową tablicę, która będzie kopią tablicy tymczasowej – rozmiar tej nowej tablicy wyznaczy liczba unikalnych elementów, które znaleźliśmy w poprzednim punkcie.

Aby odpowiedzieć na pytanie czy dana wartość powinna zostać pominięta przy przepisywaniu wystarczy sprawdzić, czy poprzedzająca ją wartość jest taka sama – jeżeli tak, to znaczy, że ta wartość jest zduplikowana i należy ją pominąć. Szczególnym przypadkiem jest pierwszy element tablicy źródłowej – nie ma poprzednika, więc zawsze przepiszemy go do tymczasowej tablicy. Ten algorytm de-duplikacji zadziała, ponieważ nasza tablica będzie posortowana – w posortowanej tablicy te same (zduplikowane) wartości będą obok siebie.

Spójrzmy teraz na implementację rozwiązania tego zadania. Na początku metody `main`, w liniach `(1)` i `(2)`, pobieramy od użytkownika dwa słowa. Następnie, pobieramy znaki, z których te słowa się składają, ale w pierwszej kolejności zamieniamy w nich wszystkie litery na małe za pomocą metody `toLowerCase` (linie `(3)` i `(4)`).

Kolejnym krokiem (linie `(5)` i `(6)`) jest posortowanie obu otrzymanych tablic, abyśmy mogli usunąć z nich duplikaty. Usuwanie duplikatów z pierwszej tablicy zaczyna się w linii `(7)`, w której tworzymy tymczasową tablicę do przechowania unikalnych wartości. W pętli z linii `(8)` przechodzimy przez elementy tablicy źródłowej i przepisujemy je do tablicy tymczasowej tylko w przypadku, gdy jest to pierwszy element albo ten element jeszcze nie wystąpił (warunek z linii `(9)`). W takim przypadku zwiększamy też liczbę unikalnych wartości korzystając z postfixowego operatora `++`:

```
unikalne1[liczbaUnikalnych1++] = znaki1[i];
```

Dzięki temu po zakończeniu pętli wiemy ile jest unikalnych elementów w tablicy tymczasowej – pozostaje nam utworzyć nową tablicę, której rozmiarem będzie właśnie liczba unikalnych wartości. To zadanie oddelegujemy metodzie `Array.copyOfRange` w linii `(10)`.

Pytania_i_zadania_rozwiazania/Rozdzial_09/CzySlowaZTakichSamychLiter.java

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

public class CzySlowaZTakichSamychLiter {
  public static void main(String[] args) {
    System.out.print("Podaj pierwsze słowo: ");
    String slowo1 = pobierzSlowo(); // 1

    System.out.print("Podaj drugie słowo: ");
    String slowo2 = pobierzSlowo(); // 2

    char[] znaki1 = slowo1.toLowerCase().toCharArray(); // 3
    char[] znaki2 = slowo2.toLowerCase().toCharArray(); // 4

    Arrays.sort(znaki1); // 5
    Arrays.sort(znaki2); // 6

    char[] unikalne1 = new char[znaki1.length]; // 7

    int liczbaUnikalnych1 = 0;
    for (int i = 0; i < znaki1.length; i++) {
      if (i == 0 || znaki1[i] != znaki1[i - 1]) { // 9
        unikalne1[liczbaUnikalnych1++] = znaki1[i];
      }
    }
    unikalne1 = // 10
       Arrays.copyOfRange(unikalne1, 0, liczbaUnikalnych1);

    char[] unikalne2 = new char[znaki2.length];

    int liczbaUnikalnych2 = 0;
    for (int i = 0; i < znaki2.length; i++) {
      if (i == 0 || znaki2[i] != znaki2[i - 1]) {
        unikalne2[liczbaUnikalnych2++] = znaki2[i];
      }
    }
    unikalne2 =
       Arrays.copyOfRange(unikalne2, 0, liczbaUnikalnych2);
```

```
    if (Arrays.equals(unikalne1, unikalne2)) { // 11
      System.out.println(
          "Te słowa składają się z takich samych znaków."
      );
    } else {
      System.out.println(
          "Te słowa składają się z innych znaków."
      );
    }
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

Musimy tę samą operację wykonać dla tablicy zawierającej znaki drugiego słowa. Gdy mamy obie tablice bez duplikatów, możemy je porównać w linii `(11)` i wypisać na ekran odpowiedni komunikat.

Prześledźmy operacje wykonywane w tym programie, gdy użytkownik poda słowa „MAMA” i „ma”:

- `(1)` i `(2)` – zmienne `slowo1` i `slowo2` będą miały wartości `"MAMA"` i `"ma"`,
- `(3)` i `(4)` – zmiennym `znaki1` i `znaki2` zostaną przypisane tablice:

```
znaki1 -> { 'm', 'a', 'm', 'a' }
znaki2 -> { 'm', 'a' }
```

- `(5)` i `(6)` – tablice `znaki1` i `znaki2` zostaną posortowane

```
znaki1 -> { 'a', 'a', 'm', 'm' }
znaki2 -> { 'a', 'm' }
```

- `(7)` – do zmiennej `unikalne1` zostanie przypisana nowa czteroelementowa tablica, która będzie wypełniona domyślnymi wartościami typu `char`,
- pętla z warunkiem z linii `(9)` – w ramach wykonania tej pętli do tablicy `unikalne1` zostaną przepisane unikalne znaki z tablicy `znaki1`, tzn. po zakończeniu tej pętli tablica, na którą wskazuje zmienna `unikalne1`, będzie zawierała następujące elementy (`'\u0000'` to domyślna wartość typu `char`):

```
unikalne1 -> { 'a', 'm', '\u0000', '\u0000' }
```

- `(10)` – do zmiennej `unikalne1` zostanie przypisana nowa tablica, która będzie kopią tablicy, na którą zmienna `unikalne1` obecnie wskazuje. Ta kopia będzie zawierała tylko tyle elementów, ile unikalnych wartości zostało

znalezionych podczas przepisywania elementów z tablicy `znaki1`. Po tej operacji zmienna `unikalne1` będzie wskazywała na następującą tablicę:

```
unikalne1 -> { 'a', 'm' }
```

- operacja de-duplikacji zostanie wykonana dla drugiego słowa – jej wynikiem będzie zmienna `unikalne2`, która będzie wskazywała na tablicę:

```
unikalne2 -> { 'a', 'm' }
```

- `(11)` – obie tablice z niepowtarzającymi się, posortowanymi znakami zostaną porównane, a na ekran zostanie wypisany odpowiedni komunikat.

Kilka przykładowych uruchomień tego programu:

```
Podaj pierwsze słowo: mama
Podaj drugie słowo: MA
Te słowa składają się z takich samych znaków.
```

```
Podaj pierwsze słowo: kajak
Podaj drugie słowo: jak
Te słowa składają się z takich samych znaków.
```

```
Podaj pierwsze słowo: program
Podaj drugie słowo: gram
Te słowa składają się z innych znaków.
```

9.2.9. Liczby na przemian

Napisz program, w którym zdefiniujesz tablicę liczb całkowitych. Program powinien następnie posortować tę tablicę w następujący sposób: pierwszym elementem powinna być najmniejsza liczba w tej tablicy. Drugim elementem powinna być największa liczba w tej tablicy. Trzecim elementem powinna być druga-najmniejsza liczba w tej tablicy, czwartym – druga-największa itd.

Dla przykładu, jeżeli tablica zawiera elementy:

```
10, 1, 7, 13, 42
```

to ta tablica powinna zostać finalnie posortowana do następującej postaci:

```
1, 42, 7, 13, 10
```

W rozwiązania tego zadania posłużymy się pomocniczą, tymczasową tablicą, do której wstawimy element z tablicy źródłowej w takiej kolejności, jak opisana w treści zadania. Potem tę tymczasową tablicę przypiszemy do źródłowej zmiennej-tablicy.

Aby osiągnąć założenia tego zadania, musimy wiedzieć jaka jest najmniejsza i największa liczba w tej tablicy. Musimy też znać drugą najmniejszą i drugą największą liczbę itd. To wstępne wymaganie możemy w łatwy sposób osiągnąć, jeżeli posortujemy źródłową tablicę – potem zostanie nam już tylko przepisanie jej elementów w odpowiedniej kolejności do pomocniczej tablicy. Spójrzmy na proponowaną implementację rozwiązania tego zadania.

Na początku metody `main` definiujemy przykładową tablicę `liczby`, a także zmienną długość, która będzie przechowywać długość tej tablicy – ta wartość będzie nam potrzebna w kilku miejscach w dalszej części programu.

W linii `(1)` sortujemy tablicę źródłową, a w linii `(2)` tworzymy tymczasową tablicę, do której wstawimy elementy w kolejności zgodnej z wymaganiami tego zadania.

Pytania_i_zadania_rozwiazania/Rozdzial_09/LiczbyNaPrzemian.java

```
import java.util.Arrays;

public class LiczbyNaPrzemian {
  public static void main(String[] args) {
    int[] liczby = { 10, 1, 7, 13, 42 };
    int dlugosc = liczby.length;

    Arrays.sort(liczby); // 1
    int[] tymczasowaTablica = new int[dlugosc]; // 2

    // 3
    for (int i = 0, j = 0; i < dlugosc / 2; i++, j += 2) {
      tymczasowaTablica[j] = liczby[i]; // 4
      tymczasowaTablica[j + 1] =
          liczby[dlugosc - i - 1]; // 5
    }

    if (dlugosc % 2 == 1) { // 6
      tymczasowaTablica[dlugosc - 1] = liczby[dlugosc / 2];
    }

    liczby = tymczasowaTablica;

    System.out.println(
        "Zawartość tablicy liczby: " +
        Arrays.toString(liczby)
    );
  }
}
```

Ustawianie elementów w odpowiedniej kolejności dzieje się w pętli `for`, która zaczyna się w linii `(3)`. Zwróć uwagę, że używamy w niej dwóch różnych zmiennych: `i` oraz `j`. Pierwsza służy do wyznaczania aktualnego elementu tablicy źródłowej, natomiast druga wskazuje element z pomocniczej tablicy, do której ma zostać wstawiona „aktualna" najmniejsza liczba. W każdym obiegu tej pętli do pomocniczej tablicy wstawimy nie jedną, lecz dwie wartości: w pierwszym obiegu będą to najmniejsza i największa liczba, w drugim obiegu druga najmniejsza i druga największa itd. – jest to zapewnione przez posortowanie źródłowej tablicy `liczby`, którą to operację wykonaliśmy w linii `(1)`. Najmniejsza liczba wstawiana jest jako pierwsza, pod indeksem wyznaczanym przez zmienną `j`, natomiast największa – pod następnym indeksem, czyli `j + 1`. Do tych elementów wstawiamy, odpowiednio, zmienną z „lewego" końca tablicy źródłowej (tego pod indeksem `i`), oraz z „prawego" końca tablicy źródłowej (tego pod indeksem `dlugosc - i - 1`).

W związku z tym, że w każdym obiegu przepisujemy dwa elementy z tablicy źródłowej, to musimy przejść tylko przez połowę jej elementów – stąd warunek pętli:

```
i < dlugosc / 2
```

Dodatkowo, skoro do tablicy tymczasowej wstawiamy dwa elementy, to musimy wartość zmiennej `j` w każdym obiegu pętli zwiększyć o dwa, a nie o jeden:

```
j += 2
```

W ten sposób w kolejnych obiegach pętli będziemy wstawiać do tablicy tymczasowej kolejne pary liczb: najmniejszą i największą.

Po zakończeniu pętli musimy jeszcze sprawdzić w linii `(6)`, czy tablica źródłowa ma nieparzystą liczbę elementów – jeżeli tak, to musimy ten element wstawić do tablicy tymczasowej – jest to spowodowane tym, że ten element nie ma pary i nie zostanie obsłużony (przepisany) w pętli `for`.

Na końcu metody `main` przypisujemy do zmiennej-tablicy `liczby` nowo utworzoną tablicę z elementami ustawionymi zgodnie z opisem zadania, po czym wypisujemy jej zawartość na ekran:

```
Zawartość tablicy liczby: [1, 42, 7, 13, 10]
```

9.2.10. Usuwanie wszystkich wystąpień liczby z tablicy

Napisz program, w którym utworzysz tablicę liczb całkowitych. Następnie, napisz kod, który usunie z tej tablicy wszystkie wystąpienia pewnej liczby. Dla przykładu, jeżeli tablica ma następujące elementy:

```
10, 5, 99, 100, 10, -42, 10
```

i będziemy chcieli z niej usunąć liczbę `10`, to po usunięciu elementów mających wartość `10` nasza tablica powinna wyglądać następująco:

```
5, 99, 100, -42
```

Możemy to zadanie wykonać na dwa sposoby – omówimy każdy z nich.

Z racji tego, że rozmiaru raz utworzonej tablicy nie można zmienić, aby wykonać to zadanie będziemy potrzebowali dodatkowej tablicy, do której przepiszemy ze źródłowej tablicy wszystkie elementy poza tymi, które mamy usunąć. Będzie to pierwszy krok do wykonania naszego zadania, ponieważ po wykonaniu tej operacji musimy utworzyć jeszcze jedną tablicę, której rozmiar będzie równy liczbie elementów, które pozostały po usunięciu wskazanej wartości – do tego celu świetnie nada się metoda `Arrays.copyOfRange`.

Spójrzmy na pierwsze z możliwych rozwiązań. Na początku metody `main` zdefiniowana jest przykładowa tablica `liczby` oraz wartość, która ma zostać usunięta, zapisana w pomocniczej zmiennej `liczbaDoUsuniecia`.

W linii `(1)` definiujemy pomocniczą tablicę, do które przepiszemy elementy ze źródłowej tablicy, ale z pominięciem wystąpień liczby do usunięcia. Elementy przepisujemy w pętli, która zaczyna się w linii `(2)`. Jeżeli aktualna liczba jest inna niż ta, która ma zostać usunięta, to przepisujemy ją do tymczasowej tablicy w linii `(3)`.

Indeksy w tymczasowej i źródłowej tablicy przestaną sobie odpowiadać w momencie, gdy trafimy na pierwszy element, który ma zostać usunięty. Dlatego indeks w tablicy tymczasowej wyznaczany jest przez dodatkową zmienną `indeksWTymczasowej` – wartość tej zmiennej będzie zmieniana tylko wtedy, gdy przepiszemy do tablicy tymczasowej wartość z tablicy źródłowej.

Na końcu programu pozostaje utworzyć nową tablicę, której rozmiar będzie równy rozmiarowi oryginalnej tablicy pomniejszony o liczbę elementów, które zostały z niej „usunięte”. Ten rozmiar wyznacza wartość zmiennej `indeksWTymczasowej` po zakończeniu pętli. Po utworzeniu nowej tablicy wypisujemy jej zawartość na ekran.

Pytania_i_zadania_rozwiazania/Rozdzial_09/UsuwanieWystapienLiczby.java

```
import java.util.Arrays;

public class UsuwanieWystapienLiczby {
  public static void main(String[] args) {
    int[] liczby = { 10, 5, 99, 100, 10, -42, 10 };
    int liczbaDoUsuniecia = 10;

    int[] tymczasowaTablica = new int[liczby.length]; // 1
```

```
    int indeksWTymczasowej = 0;
    for (int i = 0; i < liczby.length; i++) { // 2
      if (liczby[i] != liczbaDoUsuniecia) {
        tymczasowaTablica[indeksWTymczasowej++] = liczby[i]; // 3
      }
    }

    liczby = Arrays.copyOfRange(
        tymczasowaTablica, 0, indeksWTymczasowej
    );

    System.out.printf(
        "Zawartość tablicy po usunięciu liczby %d: %s",
        liczbaDoUsuniecia, Arrays.toString(liczby)
    );
  }
}
```

Innym sposobem na rozwiązanie tego zadania jest sprawdzenie ile razy liczba do usunięcia występuje w tablicy źródłowej zanim przystąpimy do przepisania jej elementów do tablicy tymczasowej – dzięki temu od razu będziemy znali rozmiar wynikowej (tzn. z „usuniętymi" elementami) tablicy.

W linii `(1)` zliczamy ile razy liczba do usunięcia występuje w tablicy źródłowej. Ta informacja posłuży nam do wyznaczenia rozmiaru tymczasowej tablicy w linii `(2)`.

W drugi pętli przepisujemy elementy w taki sam sposób, jak w poprzedniej wersji tego programu, jednak tym razem użyłem pętli `for`-each. Po zakończeniu pętli do zmiennej-tablicy `liczby` przypisujemy tablicę, na którą wskazuje zmienna `tymczasowaTablicy` – dzięki temu zmienna `liczby` będzie wskazywać na tablicę, która nie zawiera elementów, które miały zostać usunięte.

Pytania_i_zadania_rozwiazania/Rozdzial_09/UsuwanieWystapienLiczbyV2.java

```
import java.util.Arrays;

public class UsuwanieWystapienLiczbyV2 {
  public static void main(String[] args) {
    int[] liczby = { 10, 5, 99, 100, 10, -42, 10 };
    int liczbaDoUsuniecia = 10;

    int ileDoUsuniecia = 0;
    for (int liczba : liczby) { // 1
      if (liczba == liczbaDoUsuniecia) {
        ileDoUsuniecia++;
      }
    }
```

```
    int[] tymczasowaTablica =
        new int[liczby.length - ileDoUsuniecia]; // 2

    int indeksWTymczasowej = 0;
    for (int liczba : liczby) {
      if (liczba != liczbaDoUsuniecia) {
        tymczasowaTablica[indeksWTymczasowej++] = liczba;
      }
    }

    liczby = tymczasowaTablica;

    System.out.printf(
        "Zawartość tablicy po usunięciu liczby %d: %s",
        liczbaDoUsuniecia, Arrays.toString(liczby)
    );
  }
}
```

Przykładowe uruchomienie programu:

```
Zawartość tablicy po usunięciu liczby 10: [5, 99, 100, -42]
```

METODY

10.1. Pytania

1. Spójrz na poniższą metodę i odpowiedz na pytania:
    a. Co robi ta metoda?
    b. Jakie argumenty przyjmuje i co zwraca?
    c. Czy i w jaki sposób ta metoda mogłaby być lepiej napisana?

```
public static int wykonajDzialanie(int x, int y) {
  return x / y;
}
```

Ta metoda zwraca wynik dzielenia liczb przesłanych jako jej argumenty. Argumentami tej metody są `x` oraz `y` – dwie wartości typu `int`. Ta metoda powinna sprawdzać, czy liczba `y` jest równa `0` – w takim przypadku dzielenie nie powinno zostać wykonane. Pytanie tylko: co wtedy zwrócić? Moglibyśmy *rzucić wyjątek*. Wyjątki są tematem kolejnego tomu *„Javy od podstaw”*.

---

2. Co zrobić, aby użyć metody (wywołać ją)?

Aby użyć metody, czyli ją wywołać, należy napisać jej nazwę oraz nawiasy `()`, w których należy umieścić argumenty metody (o ile jakieś przyjmuje).

---

3. Które z poniższych są poprawnymi nazwami metod?
    a. `_mojaMetoda`
    b. `zapiszUstawienia`
    c. `zapiszKosztW$`
    d. `5NajlepszychOfert`
    e. `pobierz5OstatnichZamowien`

Poprawne nazwy metod to `_mojaMetoda`, `zapiszUstawienia`, `zapiszKosztW$`, oraz `pobierz5OstatnichZamowien`. Jedyną nieprawidłową nazwą jest `5NajlepszychOfert`, ponieważ zaczyna się od liczby.

---

4. Czy kolejność definiowania metod w pliku z kodem źródłowym ma znaczenie?

Dla kompilatora kolejność metod nie ma znaczenia, tzn. mogą one być zdefiniowane w dowolnej kolejności. Jednakże, metody można umieścić w kodzie źródłowym w takiej kolejności, aby ułatwić osobie czytającej nasz kod zrozumienie go i utrzymanie.

---

5. Czym są zmienne lokalne?

Zmienne lokalne to zmienne zdefiniowane w metodach. Są one dostępne tylko w metodach, w których zostały zdefiniowane.

---

6. Czy możemy z metody `abc` odwołać się do zmiennej lokalnej zdefiniowanej w metodzie `xyz`?

Nie, zmienne lokalne są niedostępne poza metodami, w których zostały zdefiniowane.

---

7. Jaki jest zakres życia zmiennych lokalnych?

Zmienne lokalne „żyją" w metodach, w których zostały zdefiniowane – gdy metoda się kończy, te zmienne przestają istnieć.

---

8. Czy poniższy kod jest poprawny i jak go ewentualnie naprawić?

```
public class ZakresIWywolywanieMetodPytanie1 {
  public static void main(String[] args) {
    System.out.println("Wynik: " + kwadrat);
    int kwadrat = policzKwadrat(10);
  }

  public static int policzKwadrat(int liczba) {
    return liczba * liczba;
  }
}
```

Ten program się nie skompiluje, ponieważ próbujemy użyć zmiennej `kwadrat` zanim zostanie zdefiniowana. Aby program był poprawny, należy przenieść definicję zmiennej `kwadrat` przed instrukcję `System.out.println`:

```
public class ZakresIWywolywanieMetodPytanie1 {
  public static void main(String[] args) {
    int kwadrat = policzKwadrat(10);
    System.out.println("Wynik: " + kwadrat);
  }

  public static int policzKwadrat(int liczba) {
    return liczba * liczba;
  }
}
```

---

9. Jaki będzie wynik działania poniższego programu?

```
public class ZakresIWywolywanieMetodPytanie2 {
  public static void main(String[] args) {
    policzKwadrat(10);
    System.out.println("Wynik: " + wynik);
  }

  public static void policzKwadrat(int liczba) {
    int wynik = liczba * liczba;
  }
}
```

Program się nie skompiluje, ponieważ w metodzie `main` próbujemy skorzystać z zmiennej lokalnej utworzonej w metodzie `policzKwadrat`. Zmienna `wynik` jest niedostępna poza metodą `policzKwadrat` – kompilator zgłosi błąd.

---

10. Co zobaczymy po kolei na ekranie po uruchomieniu poniższego programu?

```
public class ZakresIWywolywanieMetodPytanie3 {
  public static void main(String[] args) {
    System.out.println(
        "Witajcie! Kwadrat 10 to: " + policzKwadrat(10)
    );
    System.out.println("Policzone!");
  }
```

```
  public static int policzKwadrat(int liczba) {
     System.out.println("Liczymy kwadrat liczby " + liczba);
     return liczba * liczba;
  }
}
```

W pierwszej linii metody `main` częścią argumentu `System.out.println` jest wartość zwrócona z metody `policzKwadrat`, więc najpierw wykonane zostanie ciało metody `policzKwadrat`, a dopiero potem w metodzie `main` wypisany zostanie pierwszy komunikat. Dlatego, na ekranie zobaczymy:

```
Liczymy kwadrat liczby 10
Witajcie! Kwadrat 10 to: 100
Policzone!
```

---

11. Jak zwrócić z metody wartość?

Należy użyć słowa kluczowego `return`, po którym powinna nastąpić wartość, którą chcemy zwrócić. Na końcu powinniśmy umieścić średnik.

---

12. Czy metoda, która zwraca liczbę, może także zwrócić tekst (`String`)?

Nie, zwracać możemy wartość tylko jednego typu.

---

13. Czy metoda może nic nie zwracać, a jeśli tak, to jak to osiągnąć?

Tak, metoda może nic nie zwracać. Wtedy definiujemy zwracany przez metodę typ jako `void`.

---

14. Czy możemy użyć słowa kluczowego `return` więcej niż raz w metodzie?

Tak, słowa kluczowego `return` możemy używać wielokrotnie w metodach, np. w blokach instrukcji `if`.

---

15. Czy możemy użyć słowa kluczowego `return` w metodzie, która nic nie zwraca?

Tak, ale nie może po nim nastąpić żadna wartość – w przeciwnym razie program się nie skompiluje.

---

16. Jeżeli metoda ma zwrócić wartość typu `double`, ale nie użyjemy w niej `return`, czy kod się skompiluje?

Nie, program się nie skompiluje. Kompilator jest w stanie wykryć sytuację, w której metoda nie zwraca wartości i nie pozwoli na skompilowanie takiego kodu.

---

17. Gdzie możemy użyć wartości zwracanej przez metodę?

Wartość zwracaną przez metodę możemy użyć wszędzie tam, gdzie jest spodziewana wartość bądź wyrażenie, którego metoda może być częścią. Możemy m.in. przypisać wynik działania metody do zmiennej, użyć wyniku metody jako argumentu innej metody, lub w warunku instrukcji warunkowej itp.

---

18. Czy wartość zwrócona przez metodę musi zostać zawsze użyta?

Nie musi. Możemy zignorować wartość zwracaną przez metodę.

---

19. Czy poniższy kod jest poprawny (klasa opakowująca metody została pominięta)?

```
public static void main(String[] args) {
  String tekst = podniesDoKwadratu(16);
}

public static int podniesDoKwadratu(int liczba) {
  return liczba * liczba;
}
```

Ten fragment kodu nie jest poprawny, ponieważ wynik metody, która zwraca wartość typu `int`, próbujemy przypisać do zmiennej `tekst`, która jest typu `String`.

---

20. Które z poniższych metod są nieprawidłowe i dlaczego?

```
public static wypiszKomunikat() {
  System.out.println("Witajcie!");
}
```

Ta metoda jest nieprawidłowa, ponieważ przed jej nazwą brakuje zwracanego typu.

```
public static void x() {}
```

Ta metoda jest poprawna. Nic nie zwraca, nie przyjmuje argumentów, i nie wykonuje żadnych instrukcji.

```
public static void pobierzLiczbe() {
  return new Scanner(System.in).nextInt();
}
```

Ta metoda jest niepoprawna, ponieważ próbuje zwrócić wartość pomimo, że jej zwracany typ to `void`, czyli metoda nie powinna zwracać jakiejkolwiek wartości.

```
public static int doKwadratu(int c) {
  int wynik = c * c;
}
```

Ta metoda jest niepoprawna, gdyż nie zwraca żadnej wartości, a powinna, ponieważ typ `int` jest zdefiniowany jako typ wartości, który ta metoda powinna zwracać.

```
public static int podzielLiczby(int a, int b) {
  if (b == 0) {
    return;
  }
  return a / b;
}
```

Ta metoda jest niepoprawna, ponieważ istnieje taka ścieżka wykonania, w której metoda nie zwróci wartości. Jeżeli `b` będzie równe `0`, to metoda nie zwróci wartości – kompilator jest w stanie wykryć ten problem już na etapie kompilacji.

```
public static String pobierzLiczbe() {
  return new Scanner(System.in).nextInt();
}
```

Ta metoda jest niepoprawna, ponieważ metoda próbuje zwrócić wartość typu `int`, a definiuje, że będzie zwracać wartość typu `String`.

```
public static int ktoraNajwieksza(int a, int b, int c) {
  if (a > b) {
    if (a > c) {
      return a;
    }
  } else {
    if (c > b) {
      return c;
    } else {
      return b;
    }
  }
}
```

Ta metoda jest niepoprawna, ponieważ istnieje szansa, że nie zostanie z niej zwrócona wartość. Jeżeli `a > b` oraz `a <= c`, to metoda nie zwróci wartości. Kompilator wykryje potencjalny problem i nie skompiluje powyższego kodu. Aby metoda była poprawna, powinniśmy po instrukcji warunkowej sprawdzającej czy `a > c` dodać zwracanie wartości zmiennej `c`:

```
if (a > c) {
  return a;
} else {
  return c;
}
```

```
public static void wypiszKwadrat(int a) {
  System.out.println("Kwadrat wynosi: " + a * a);
  return;
  System.out.println("Policzone!"); // 1
}
```

Metoda nie jest poprawna, ponieważ linia `(1)` nigdy nie ma szansy na wykonanie ze względu na wcześniejsze użycie `return`. Kompilator jest w stanie to wykryć i nie pozwoli na kompilację.

```
public static int podzielLiczby(int a, int b) {
  if (b == 0) {
    return null;
  }
  return a / b;
}
```

Metoda jest niepoprawna, ponieważ wartość `null` nie może być reprezentowana przez typy prymitywne, więc z metody, która ma zwrócić wartość typu `int`, próba zwrócenia wartości `null` zakończy się błędem kompilacji.

```
public static void wypiszPowitanie {
  System.out.println("Witajcie!");
}
```

Metoda jest niepoprawna, ponieważ brakuje nawiasów `()` po nazwie metody. Nawiasy `()` są wymagane nawet w przypadku, gdy metoda nie przyjmuje żadnych argumentów.

---

21. Do czego służą argumenty metod?

Argumenty metod to dane wejściowe, które przekazujemy do metod. Argumenty są wykorzystywane przez metody do wykonania określonej operacji.

---

22. Czy metody mogą przyjmować zero argumentów?

Tak, metody mogą przyjmować zero argumentów.

---

23. Czy metodę, która przyjmuje jeden argument – liczbę typu `int` – możemy wywołać bez podania żadnego argumentu?

Nie, wszystkie argumenty metody są wymagane i nie możemy pominąć żadnego z nich podczas wywoływania metody, chociaż jest jeden szczególny przypadek – argument *varargs*. Jeżeli metoda przyjmuje taki argument, to możemy go pominąć – do metody zostanie wtedy automatycznie przekazana pusta tablica.

---

24. Czy kolejność argumentów ma znaczenie?

Tak, kolejność argumentów ma znaczenie – metoda przyjmująca `String` oraz liczbę typu `int` to inna metoda, niż metoda przyjmująca liczbę typu `int` i `String`.

---

25. Jeżeli w metodzie zmodyfikujemy argument typu prymitywnego, to czy po powrocie z tej metody wartość zmiennej użytej jako argument zachowa wartość ustawioną w wywołanej metodzie, czy będzie miała swoją oryginalną wartość?

Po zakończeniu metody zmienna użyta jako argument będzie miała swoją oryginalną wartość. Zmiany argumentów w metodach nie są propagowane w przypadku wartości typów prymitywnych.

---

26. Jak dokumentuje się metody, opisując ich działanie, parametry, zwracany typ?

Aby udokumentować metodę, stosujemy komentarze dokumentacyjne, które zaczynają się od znaków `/**` i kończą się znakami `*/`. W komentarzu dokumentacyjnym można używać specjalnych znaczników do opisu np. parametrów metody i zwracanej przez nią wartości.

---

27. Do czego służy program `Javadoc`?

To narzędzie generuje dokumentację z kodu źródłowego Java na podstawie umieszczonych w nim komentarzy dokumentacyjnych.

---

28. Dokumentując metodę, w jaki sposób wyszczególnia się jej parametry?

Za pomocą znacznika `@param`, po którym powinna nastąpić nazwa i opis parametru.

---

29. Czy pierwsze zdanie w opisie metody w komentarzu dokumentującym ma jakieś specjalne znaczenie?

Pierwsze zdanie tego opisu ma szczególne znaczenie – jest ono umieszczane obok nazwy metody na liście metod danej klasy. Pozostałe zdania opisu są zawarte w pełnym opisie metody w dalszej części dokumentacji generowanej za pomocą narzędzia `Javadoc`.

---

30. Czy poniższy kod jest poprawny?

```
public class Pytanie {
  public static void main(String[] args) {
    wypiszKomunikat;
  }

  public static void wypiszKomunikat() {
    System.out.println("Witajcie!");
  }
}
```

Ten kod nie jest poprawny, ponieważ podczas wywoływania metody należy po jej nazwie umieścić nawiasy `()`. Aby kod był poprawny, w metodzie `main` linia `wypiszKomunikat;` powinna być zastąpiona linią `wypiszKomunikat();`

---

31. Jaka wartość zostanie wypisana na ekran w poniższym programie?

```
public class Pytanie {
  public static void main(String[] args) {
    int[] tab = { 7, 8, 9 };
    metoda(tab);
    System.out.println(tab[0]);
  }

  public static void metoda(int[] tab) {
    int[] tablica = tab; // 1

    for (int i = 0; i < tab.length; i++) {
      tablica[i] = tab[i] * tab[i];
    }
  }
}
```

Na ekranie zobaczymy liczbę `49`. Zmiany wartości typów złożonych (do których tablice przynależą) są wykonywane na oryginalnym obiekcie, a nie na jego kopii. Dlaczego jednak tak się stało, skoro w linii `(1)` tworzymy nową zmienną `tablica` i przypisujemy do niej wartość argumentu `tab`? Powodem jest to, że nie tworzymy nowej tablicy, a jedynie zmienną, która na nią pokazuje. W metodzie `metoda` zarówno zmienna `tablica`, jak i argument `tab`, wskazują na tę samą tablicę – tę, która została przekazana jako argument w metodzie `main`.

---

32. Jaki będzie efekt uruchomienia poniższego programu?

```
public class Pytanie2 {
  public static void main(String[] args) {
    int[] tab = { 7, 8, 9 }; // 1

    metoda(tab);

    System.out.println(tab.length);
  }

  public static void metoda(int[] tab) {
    tab = null;
  }
}
```

Na ekranie zobaczymy liczbę `3`, czyli długość tablicy zdefiniowanej w linii `(1)`. Zmiana tego, na co pokazuje argument `tab` w metodzie `metoda`, nie ma wpływu na to, na co pokazuje zmienna `tab` zdefiniowana w linii `(1)`.

---

33. Które z poniższych sygnatur metod (pomijamy brak ciała metod) są nieprawidłowe i dlaczego?

```
public static void metoda(wiadomosc String)
```

Ta metoda jest nieprawidłowa, ponieważ typ argumentu i jego nazwa zapisane są w nieprawidłowej kolejności. Najpierw powinien zostać zdefiniowany typ argumentu, a dopiero potem jego nazwa. Poprawnie zapisany argument to:

```
String wiadomosc
```

```
public static void metoda
```

Ta metoda jest nieprawidłowa, ponieważ nie ma nawiasów `()` po swojej nazwie.

```
public static void metoda(int byte, char znak)
```

Ta metoda jest nieprawidłowa, ponieważ jako nazwy pierwszego argumentu używa nazwy zastrzeżonej w języku Java – `byte` to słowo kluczowe, nazwa jednego z typów prymitywnych.

```
public static void metoda()
```

Sygnatura tej metody jest prawidłowa – nie przyjmuje argumentów i nic nie zwraca.

```
public static void metoda(int #numerPracownika)
```

Ta metoda jest nieprawidłowa, ponieważ jej argument ma błędną nazwę. Nazwy nie mogą zawierać znaku `#`.

```
public static void metoda(string wiadomosc)
```

Ta metoda jest nieprawidłowa, ponieważ typ argumentu jest błędny – typ `string` nie istnieje w języku Java – poprawnie zapisany argument to `String wiadomosc`.

```
public static void metoda(int)
```

Ta metoda jest nieprawidłowa, ponieważ pierwszy argument nie ma nazwy.

```
public static void metoda(int liczba String wiadomosc)
```

Ta metoda jest nieprawidłowa, ponieważ brakuje znaku przecinka, który oddzieliłby od siebie argumenty tej metody.

```
public static void metoda(int _numerPracownika)
```

Ta metoda jest poprawna – nazwy w Javie mogą zaczynać się od podkreślenia.

```
public static void metoda(double pi = 3.14)
```

Ta metoda jest nieprawidłowa, ponieważ argumenty w języku Java nie mogą mieć wartości domyślnych.

---

34. Jak będzie wynik wykonania poniższego programu?

```
public class Pytanie {
  public static void main(String[] args) {
    metoda(3.14, 20);
  }

  public static void metoda(
      int liczba, double drugaLiczba
  ) {
    System.out.println("Liczba = " + liczba);
    System.out.println("Druga liczba = " + drugaLiczba);
  }
}
```

Ten program w ogóle się nie skompiluje, ponieważ argumenty przesyłane do metody `metoda` są w złej kolejności. Ta metoda jako pierwszego argumentu spodziewa się liczby całkowitej, a jako drugiego – liczby rzeczywistej. W metodzie `main` podajemy te argumenty w odwrotnej kolejności.

---

35. Do czego służą argumenty „varargs”?

Argumenty varargs pozwalają metodzie na otrzymywanie dowolnej liczby argumentów pewnego typu.

---

36. Która z poniższych definicji metod jest poprawna, a która nie i dlaczego?

```
public static int pewnaMetoda(String... slowa, int liczba)
{
  // ...
}

public static int pewnaMetoda2(
    String... slowa, int... liczby
) {
  // ...
}
```

Żadna z tych definicji nie jest poprawna. Pierwsza metoda zawiera błąd, ponieważ argument varargs musi być zawsze ostatnim argumentem metody. Druga metoda jest niepoprawna, ponieważ metoda może mieć co najwyżej jeden argument varargs.

---

37. Czym jest przeładowywanie metod?

Przeładowanie metod to definicja metod o tych samych nazwach, ale różnych argumentach. Metody te muszą różnić się liczbą argumentów, ich typami, lub kolejnością.

---

38. Które z poniższych par sygnatur przeładowanych metod są poprawne, a które nie? Wyjaśnij, dlaczego.

```
public static void metoda(int liczba)
public static int metoda(int liczba)
```

Te metody nie są poprawnie przeładowane, ponieważ różnią się jedynie zwracanym typem, a zwracany typ nie ma znaczenia podczas przeładowania metod.

```
public static void metoda(int liczba, int drugaLiczba)
public static void metoda(int liczba)
```

Te metody są poprawnie przeładowane – mają różną liczbę argumentów.

```
public static void metoda(int liczba)
public static void metoda(int liczba)
```

Te metody nie są poprawnie przeładowane, ponieważ są identyczne – w ogóle się od siebie nie różnią.

```
public static void metoda(double liczba)
public static void metoda(int liczba)
```

Te metody są poprawnie przeładowane bo przyjmują argumenty różnych typów.

```
public static void metoda(double liczba, String tekst)
public static void metoda(String tekst, double liczba)
```

Te metody są poprawnie przeładowane, gdyż ich argumenty mają inną kolejność.

```
public static void metoda(String tekst, double liczba)
public static void metoda(String komunikat, double wartosc)
```

Te metody nie są poprawnie przeładowane, bo różnią się jedynie nazwami argumentów, a nazwy argumentów nie mają znaczenia podczas przeładowania metod.

```
public static void metoda()
public static void metoda(String komunikat)
```

Te metody są poprawnie przeładowane, ponieważ różnią się liczbą argumentów.

```
public static void metoda(String komunikat)
public static void Metoda(String komunikat)
```

Te metody są poprawne, ale nie są przeładowane, ponieważ mają różne nazwy. Pierwsza z nich nazywa się `metoda`, a druga `Metoda` – wielkość znaków ma znaczenie w języku Java, więc mamy w tym przypadku po prostu dwie różne metody.

---

39. Które z poniższych ma znaczenie podczas przeładowywania metod i pozwoli na utworzenie metod o tej samej nazwie?
    a. typ zwracanej przez metodę wartości,
    b. liczba argumentów,
    c. typy argumentów,
    d. kolejność argumentów,
    e. nazwy argumentów.

Podczas przeładowania metod znaczenie mają: b) liczba argumentów, c) typy argumentów, oraz d) kolejność argumentów.

---

40. Co to jest rekurencja?

Rekurencja to wywoływanie metody przez samą siebie.

---

41. Jaki będzie efekt uruchomienia poniższego programu?

```
public class RekurencyjnaMyslAutora {
  public static void main(String[] args) {
    wyrazNiezadowolenie();
  }

  public static void wyrazNiezadowolenie() {
    System.out.println("Weekend powinien trwać 5 dni!");
    wyrazNiezadowolenie();
  }
}
```

Ten program wypisze komunikat wiele razy na ekran, po czym działanie programu zakończy się błędem `StackOverflowError`, ponieważ wysycimy miejsce na stosie przez zbyt wiele rekurencyjnych wywołań metody `wyrazNiezadowolenie`.

---

42. Jak przesłać do programu argumenty?

Należy podczas uruchamiania programu wyszczególnić je po podaniu nazwy klasy, którą chcemy uruchomić. W poniższym przykładzie do programu MojProgram przekazywane są trzy argumenty: `Ala`, `ma`, oraz `kota`:

```
> java MojProgram Ala ma kota
```

---

43. Jak przesłać do programu argument składający się z kilku słów?

Należy ująć go w cudzysłów:

```
> java MojProgram "Ala ma kota"
```

---

44. Jak odnieść się w programie do przekazanych do niego argumentów?

Argumenty przekazane do programu zawarte są w tablicy `args`, która jest argumentem metody `main`.

10.2. Przeliczanie długości na metry

W przykładzie „Zamiana długości w podanej jednostce na metry” na stronie 503 (podrozdział 10.3.8) pierwszego tomu *„Javy od podstaw”* znajdowało się dodatkowe zadanie – dodanie do tego programu sprawdzania podanej przez użytkownika jednostki – jeżeli nie zostanie rozpoznana, to program powinien poinformować o tym fakcie użytkownika.

Po uwzględnieniu opisanego wymagania ten program wygląda jak poniżej. Sprawdzanie poprawności jednostki podanej przez użytkownika znajduje się w linii `(1)`. Odpowiedzialna jest za to metoda `czyPoprawnaJednostka` zdefiniowana w linii `(3)`. Jeżeli jednostka będzie nieprawidłowa, to poinformujemy o tym fakcie użytkownika w linii `(2)`.

Pytania_i_zadania_rozwiazania/Rozdzial_10/PrzeliczanieDlugosciNaMetrySprawdzJednostke.java

```
import java.util.Scanner;

public class PrzeliczanieDlugosciNaMetrySprawdzJednostke {
  public static void main(String[] args) {
    System.out.print(
        "Podaj wartość i jednostkę oddzielone spacją: "
    );

    String dane = pobierzLinieTekstu();

    String[] wartoscIJednostka = dane.split(" ");

    if (wartoscIJednostka.length != 2) {
      System.out.println("Nieprawidłowy format danych.");
    } else {
      int wartosc = Integer.parseInt(wartoscIJednostka[0]);
      String jednostka = wartoscIJednostka[1].toLowerCase();

      if (czyPoprawnaJednostka(jednostka)) { // 1
        double wartoscWMetrach =
            zamienNaMetry(wartosc, jednostka);

        System.out.printf(
            "%d %s w metrach to: %.2f",
            wartosc, jednostka, wartoscWMetrach
        );
      } else {
        System.out.println( // 2
            "Nieznana jednostka: " + jednostka
        );
```

```
      }
    }
  }

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

  public static boolean czyPoprawnaJednostka( // 3
      String jednostka
  ) {
    return "km".equals(jednostka) || "mi".equals(jednostka) ||
        "ft".equals(jednostka) || "y".equals(jednostka) ||
        "in".equals(jednostka);
  }

  public static double zamienNaMetry(
      int wartosc, String jednostka
  ) {
    return switch (jednostka) {
      case "km" -> wartosc * 1000;
      case "mi" -> wartosc * 1609.344; // mile
      case "ft" -> wartosc * 0.30480; // stopy
      case "y" -> wartosc * 0.9144; // jardy
      case "in" -> wartosc * 0.0254; // cale
      default -> wartosc;
    };
  }
}
```

Dwa przykładowe uruchomienia tego programu:

```
Podaj wartość i jednostkę oddzielone spacją: 1 N
Nieznana jednostka: n
```

```
Podaj wartość i jednostkę oddzielone spacją: 3 km
3 km w metrach to: 3000,00
```

10.3. Zadania

10.3.1. Zadania „na rozgrzewkę”

10.3.1.1. Metoda wypisująca napis Witajcie!

Napisz metodę, która wypisuje na ekran tekst `Witajcie!`, i użyj jej w metodzie `main`.

Rozwiązanie tego zadania jest podobne do pierwszego programu, jaki napisaliśmy w ramach nauki Javy w drugim rozdziale „Javy od podstaw”, jednak tym razem oddelegowujemy wypisanie komunikatu na ekran do metody `wypiszPowitanie`:

Pytania_i_zadania_rozwiazania/Rozdzial_10/MetodaWypisujacaPowitanie.java

```
public class MetodaWypisujacaPowitanie {
  public static void main(String[] args) {
    wypiszPowitanie();
  }

  public static void wypiszPowitanie() {
    System.out.println("Witajcie!");
  }
}
```

Po uruchomieniu tego programu na ekranie zobaczymy komunikat:

```
Witajcie!
```

10.3.1.2. Metoda odejmująca dwie liczby

Napisz metodę, która wypisuje na ekran wynik odejmowania dwóch przesłanych do niej liczb, a potem użyj jej w metodzie `main`. Przykład wypisywanego komunikatu:

```
2 - 5 = -3
```

Do wypisania ładnie sformatowanego komunikatu dla użytkownika użyjemy metody `printf` wewnątrz metody `wypiszWynikOdejmowania`. Znaczniki `%d` zostaną zastąpione, odpowiednio, pierwszą liczbą, drugą liczbą, oraz wynikiem ich odejmowania:

Pytania_i_zadania_rozwiazania/Rozdzial_10/MetodaOdejmujacaDwieLiczby.java

```
public class MetodaOdejmujacaDwieLiczby {
  public static void main(String[] args) {
    wypiszWynikOdejmowania(10, 10);
    wypiszWynikOdejmowania(-5, 10);
    wypiszWynikOdejmowania(2, 5);
    wypiszWynikOdejmowania(20, -10);
  }

  public static void wypiszWynikOdejmowania(int x, int y) {
    System.out.printf("%d - %d = %d\n", x, y, x - y);
  }
}
```

Ze względu na to, że metoda `printf` nie dodaje automatycznie na koniec linii znaku nowej linii, to jest on (`\n`) dodany w wypisywanym komunikacie – dzięki temu kolejne linie tekstu będą na ekranie w osobnych liniach. Po uruchomieniu tego programu na ekranie zobaczymy:

```
10 - 10 = 0
-5 - 10 = -15
2 - 5 = -3
20 - -10 = 30
```

10.3.1.3. Metoda wypisująca gwiazdki

Napisz metodę, która wypisze podaną liczbę gwiazdek (znak `*`) na ekran.

W rozwiązaniu tego zadania w metodzie `wypiszGwiazdki` skorzystamy z pętli `for`, aby wyświetlić na ekran tyle gwiazdek, ile wskazano w argumencie `ile`:

Pytania_i_zadania_rozwiazania/Rozdzial_10/MetodaWypisujacaGwiazdki.java

```
public class MetodaWypisujacaGwiazdki {
  public static void main(String[] args) {
    wypiszGwiazdki(10);
  }

  public static void wypiszGwiazdki(int ile) {
    for (int i = 0; i < ile; i++) {
      System.out.print("*");
    }
  }
}
```

Po uruchomieniu tego programu na ekranie zobaczymy:

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

10.3.2. Zadania średniej trudności

10.3.2.1. Metoda czyPalindrom

Napisz metodę, która odpowiada na pytanie, czy podany string jest palindromem. Palindromy to słowa, które są takie same czytane od początku i od końca, np. `kajak`. Dla przykładu, dla argumentu `"kajak"` (a także `"Kajak"`) metoda powinna zwrócić `true`, a dla argumentu `"kot"` – wartość `false`. Dla argumentu będącego nullem lub pustym stringiem metoda także powinna zwrócić `false`.

W *„Javie od podstaw”* w przykładzie „Czy słowo jest palindromem?” (w rozdziale o pętlach) napisaliśmy program, który pobierał od użytkownika słowo i sprawdzał, czy jest ono palindromem. W rozwiązaniu tego zadania możemy skorzystać z użytego tam fragment kodu – umieścimy go w metodzie o nazwie `czyPalindrom`:

Pytania_i_zadania_rozwiazania/Rozdzial_10/MetodaCzyPalindrom.java

```
public class MetodaCzyPalindrom {
  public static void main(String[] args) {
    System.out.println(
        "Czy kajak to palindrom? " + czyPalindrom("kajak")
    );
    System.out.println(
        "Czy Kajak to palindrom? " + czyPalindrom("Kajak")
    );
    System.out.println(
        "Czy Anna to palindrom? " + czyPalindrom("Anna")
    );
    System.out.println(
        "Czy kot to palindrom? " + czyPalindrom("kot")
    );
    System.out.println(
        "Czy 'null' to palindrom? " + czyPalindrom(null)
    );
    System.out.println(
        "Czy pusty string to palindrom? " + czyPalindrom("")
    );
  }

  public static boolean czyPalindrom(String slowo) {
    if (slowo == null || slowo.length() == 0) { // 1
      return false;
    }
```

```
    int dlugoscSlowa = slowo.length();
    String slowoMaleLitery = slowo.toLowerCase(); // 2

    for (int i = 0; i < dlugoscSlowa / 2; i++) {
      if (
          slowoMaleLitery.charAt(i) !=
          slowoMaleLitery.charAt(dlugoscSlowa - 1 - i)
      ) {
        return false; // 3
      }
    }

    return true; // 4
  }
}
```

Wymagania z tego zadania różnią się nieco od tych z przykładu „Czy słowo jest palindromem?".

Po pierwsze, musimy uwzględnić przypadki, gdy przekazany do metody argument jest nullem lub pustym stringiem – zgodnie z opisem zadania w takim przypadku mamy zwrócić wartość `false`. To sprawdzenie wykonujemy w linii `(1)`.

Po drugie, powinniśmy traktować małe i wielkie litery jako takie same – aby osiągnąć ten cel użyjemy metody `toLowerCase` w linii `(2)`, która zwróci słowo ze wszystkimi literami zamienionymi na małe litery.

Ponadto, nie musimy korzystać z instrukcji `break` – jeżeli znajdziemy różnicę w literach na odpowiadających sobie pozycjach od początku i końca słowa, to będzie to oznaczało, że słowo nie jest palindromem – możemy zakończyć działanie metody i zwrócić wartość `false` za pomocą słowa kluczowego `return` (linia `(3)`). Jeżeli przejdziemy przez wszystkie pary liter i nie znajdziemy różnicy, to zwrócimy z metody wartość `true` (linia `(4)`) – będzie to świadczyło o tym, że przekazane jako argument słowo jest palindromem.

Po uruchomieniu tego programu na ekranie zobaczymy:

```
Czy kajak to palindrom? true
Czy Kajak to palindrom? true
Czy Anna to palindrom? true
Czy kot to palindrom? false
Czy 'null' to palindrom? false
Czy pusty string to palindrom? false
```

10.3.2.2. Nazwa w Camel Case z dowolnej liczby słów

Napisz metodę, która przyjmie dowolną liczbę wartości typu `String` i stworzy z nich nazwę zapisaną w standardzie Camel Case (pierwsze słowo powinno składać się z małych liter, a każde następne powinno być do niego doklejone z pierwszą literą zamienioną na wielką literę). Skorzystaj z mechanizmu „varargs". Jeżeli do metody przesłany zostanie `null` lub pusta tablica, metoda powinna zwrócić pusty string. Jeżeli któraś z wartości w tablicy będzie nullem lub pustym stringiem, to powinna zostać pominięta.

Przykładowe uruchomienia i spodziewane wyniki:

```
zapiszWCamelCase("Moja", "NAZWA") -> "mojaNazwa"
zapiszWCamelCase(null); -> ""
zapiszWCamelCase(""); -> ""
zapiszWCamelCase(null, "Moja", "zmienna") -> "mojaZmienna"
zapiszWCamelCase("Moja", "", null, "NAZWA") -> "mojaNazwa"
zapiszWCamelCase("Moja", "inna", null, "NAZWA") ->
"mojaInnaNazwa"
```

Przygotuj komentarz dokumentacyjny, w którym opiszesz jakie zadanie wykonuje Twoja metoda, jak obsługuje wartości `null` i puste stringi, jakie argumenty przyjmuje i co zwraca.

W metodzie `main` proponowanego rozwiązania metoda `zapiszWCamelCase` jest wywoływana kilka razy dla różnych argumentów aby sprawdzić, czy nulle i puste stringi są obsługiwane w odpowiedni sposób:

Pytania_i_zadania_rozwiazania/Rozdzial_10/StworzNazweWCamelCase.java

```
public class StworzNazweWCamelCase {
  public static void main(String[] args) {
    // oczekiwany wynik: "mojaNazwa"
    System.out.println(zapiszWCamelCase("Moja", "NAZWA"));

    // oczekiwany wynik: ""
    System.out.println(zapiszWCamelCase(null));

    // ""
    System.out.println(zapiszWCamelCase(""));

    // "mojaZmienna"
    System.out.println(
        zapiszWCamelCase(null, "Moja", "zmienna")
    );
```

```
    // "mojaNazwa"
    System.out.println(
        zapiszWCamelCase("Moja", "", null, "NAZWA")
    );

    // "mojaInnaNazwa"
    System.out.println(
        zapiszWCamelCase("Moja", "inna", null, "NAZWA")
    );
  }

  /**
   * Zapisuje podane słowa w formie standardu Camel Case.
   * W tym standardzie pierwsze słowo jest zapisane małymi
   * literami, a każde kolejne jest do niego doklejone
   * z pierwszą literą zamienioną na wielką literę,
   * natomiast pozostałe litery są małe.
   * Metoda pomija nulle i puste słowa.
   *
   * @param slowa Słowa, z których ma być stworzona nazwa
   *              zapisana w camelCase. Nulle i puste
   *              stringi są pomijane.
   * @return   Nazwa zapisana w Camel Case z podanych słów.
   *           Jeżeli przekazano nullową lub pustą tablicę
   *           ze słowami, zwrócony zostanie pusty string.
   */
  public static String zapiszWCamelCase(String... slowa) {
    if (slowa == null || slowa.length == 0) {
      return "";
    }
    String nazwaWCamelCase = "";

    for (String slowo : slowa) { // 1
      if (slowo == null || slowo.length() == 0) {
        continue; // 2
      }

      if (nazwaWCamelCase.length() == 0) { // 3
        nazwaWCamelCase += slowo.toLowerCase(); // 4
      } else {
        nazwaWCamelCase +=
            Character.toUpperCase(slowo.charAt(0)) +
            slowo.substring(1).toLowerCase(); // 5
      }
    }

    return nazwaWCamelCase;
  }
}
```

Metoda `zapiszWCamelCase` przyjmuje jeden argument typu varargs, dzięki czemu możemy wysłać do niej dowolną liczbę wartości – w tym przypadku typu `String`.

Na początku tej metody sprawdzamy, czy przekazana tablica jest nullem lub jest pusta – w takim przypadku, zgodnie z wymaganiami zadania, zwrócimy pusty string i zakończymy działanie metody.

W przeciwnym razie, pętla zaczynająca się w linii `(1)` zbuduje wynikową nazwę zapisaną w standardzie Camel Case. Jeżeli aktualnie przetwarzany fragment nazwy, czyli string zapisanych w zmiennej `slowo`, będzie nullem lub pustym stringiem, to pominiemy ten obieg pętli dzięki użyciu instrukcji `continue` w linii `(2)`. Jeżeli słowo jest poprawne, to „dokleimy" je do budowanej nazwy, której kolejne elementy przechowujemy w zmiennej `nazwaWCamelCase`.

Pierwsze słowo w nazwach zapisanych w standardzie Camel Case jest szczególne, bo jest ono zapisane małą literą – dlatego w linii `(3)` sprawdzamy, czy utworzona dotychczas nazwa jest pusta – oznacza to, że właśnie przetwarzamy pierwsze słowo, które będzie wchodziło w skład tej nazwy, więc dokleimy do niej to słowo ze wszystkimi literami zamienionymi na małe litery dzięki metodzie `toLowerCase`, którą używamy w linii `(4)`. W przeciwnym razie, tzn. gdy nie jest to pierwsze słowo budowanej nazwy, dokleimy je z pierwszą literą zamienioną na wielką literę (linia `(5)`). Na końcu metody zwracamy utworzoną nazwę.

Przed metodą, zgodnie z wymaganiami zadania, znajduje się obszerny komentarz dokumentacyjny, który wyjaśnia: jakie zadanie wykonuje metoda, jakie argumenty przyjmuje, i jak obsługuje wartości `null` i puste stringi, a także co zwraca.

Poniżej widać efekt uruchomienia tego programu. Wyświetlone są dwie puste linie, ponieważ w tych przypadkach do metody `zapiszWCamelCase` wysłane zostały, odpowiednio, `null` i pojedynczy pusty string:

```
mojaNazwa


mojaZmienna
mojaNazwa
mojaInnaNazwa
```

10.3.2.3. Metody zamieniające tablicę na string

Napisz dwie metody `arrayToString`. Pierwsza powinna przyjmować jako argument tablicę liczb całkowitych, a druga – tablicę stringów. Każda z metod powinna zwracać jej argument „zamieniony" na formę tekstową w formacie: `[element1, element2, ..., elementN]` – tak samo, jak robią to różne wersje metody `Arrays.toString`. Jeżeli przekazana zostanie pusta tablica, metody powinny zwrócić tekst `[]`, a w przypadku nullowego argumentu – tekst `null`.

Przykładowe uruchomienia i spodziewane wyniki:

```
int[] tab = null;
arrayToString(tab); -> "null"
arrayToString(new int[0]); -> "[]"
arrayToString(new int[] { 7, 13, 77 }); -> "[7, 13, 77]"

String[] tabStr = null;
arrayToString(tabStr); -> "null"
arrayToString(new int[0]); -> "[]"
arrayToString(new String[] { "Ala", "ma", null }) ->
  "[Ala, ma, null]"
```

W rozwiązaniu tego zadania skorzystamy z mechanizmu przeładowania metod, ponieważ będziemy mieli dwie metody o tej samej nazwie, ale różnych argumentach – pierwsza będzie przyjmować tablicę stringów, a druga – tablicę wartości typu `int`.

W metodzie `main` w proponowanym rozwiązaniu wielokrotnie wywołujemy obie metody z różnymi parametrami aby sprawdzić, czy poprawnie obsługują: argument będący nullem, pustą tablicę, a także (w przypadku metody przyjmującej wartości typu `String`) sytuację, w której w tablicy znajduje się `null`:

Pytania_i_zadania_rozwiazania/Rozdzial_10/ZamienTabliceNaString.java

```
public class ZamienTabliceNaString {
  public static void main(String[] args) {
    int[] nullowaTablica = null;
    int[] pustaTablica = new int[0];
    int[] tablicaZLiczbami = { 7, 13, 42, 99 };

    System.out.println(arrayToString(nullowaTablica));
    System.out.println(arrayToString(pustaTablica));
    System.out.println(arrayToString(tablicaZLiczbami));

    String[] nullowaTablicaStr = null;
    String[] pustaTablicaStr = new String[0];
    String[] tablicaZeSlowami = { "Ala", "ma", null };

    System.out.println(arrayToString(nullowaTablicaStr));
    System.out.println(arrayToString(pustaTablicaStr));
    System.out.println(arrayToString(tablicaZeSlowami));
  }
```

```
  public static String arrayToString(int[] liczby) {
    if (liczby == null) { // 1
      return "null";
    }

    String wynik = "["; // 2

    for (int i = 0; i < liczby.length; i++) { // 3
      if (i < liczby.length - 1) { // 4
        wynik += liczby[i] + ", ";
      } else {
        wynik += liczby[i];
      }
    }

    return wynik + "]"; // 5
  }

  public static String arrayToString(String[] slowa) {
    if (slowa == null) { // 1
      return "null";
    }

    String wynik = "["; // 2

    for (int i = 0; i < slowa.length; i++) { // 3
      if (i < slowa.length - 1) { // 4
        wynik += slowa[i] + ", ";
      } else {
        wynik += slowa[i];
      }
    }

    return wynik + "]"; // 5
  }
}
```

W obu metodach `arrayToString` sprawdzamy na początku, czy przekazany argument jest nullem (linie `(1)`) – jeżeli tak, to zgodnie z wymaganiami zadania zwracamy string `"null"`. W przeciwnym razie w liniach `(2)` zaczynamy budować wynikowy łańcuch tekstowy. Następnie, w pętlach zaczynających się w liniach `(3)` „doklejamy" do wyniku kolejne elementy przetwarzanej tablicy. Elementy chcemy od siebie oddzielić przecinkiem, więc doklejamy go po każdej wartości poza ostatnią – odpowiada za to warunek z linii `(4)`. Po zakończeniu pętli pozostaje nam jeszcze

dodać zamykający nawias do utworzonej tekstowej reprezentacji tablicy przekazanej jako argument (linie `(5)`).

Zwróć uwagę, że w wersji dla tablicy wartości typu `String` nie sprawdzamy w pętli, czy przetwarzany element jest nullem. Nie musimy tego robić, ponieważ nie wykonujemy na tej wartości żadnych operacji, a jedynie dodajemy ją do tworzonej tekstowej reprezentacji tablicy. Wartość `null`, gdy jest „doklejana" do stringa, zostaje „doklejona" jako string `"null"`, dzięki czemu przekazując tablicę, w której znajdują się nulle, zobaczymy w wyniku tekst `null`, co widać po uruchomieniu tego programu:

```
null
[]
[7, 13, 42, 99]
null
[]
[Ala, ma, null]
```

10.3.2.4. Rekurencyjne liczenie silni

Napisz metodę, która będzie rekurencyjnie liczyła silnię podanej liczby naturalnej. Silnia to iloczyn kolejnych liczb.

Dla przykładu, silnia liczby `5` to: `1 * 2 * 3 * 4 * 5 = 120`. Silnia liczby `0` wyjątkowo wynosi `1`. Dla nieprawidłowego argumentu, tzn. liczby mniejszej niż `0`, metoda powinna zwrócić wartość `-1`.

Metoda rekurencyjna musi posiadać pewien warunek, który spowoduje, że przestanie się sama wykonywać. W przypadku silni przerwiemy cykl wywoływania metody liczącej silnię, gdy jej argument będzie równy `0`, ponieważ jest to pierwsza liczba, dla której możemy liczyć silnię – dla tego argumentu zwrócimy liczbę `1` (zgodnie z definicją silni).

Musimy jeszcze uwzględnić przypadek, gdy do metody zostanie przesłana nieprawidłowa liczba, tzn. mniejsza od zero – zgodnie z wymaganiami tego zadania zwrócimy wtedy wartość `-1`:

Pytania_i_zadania_rozwiazania/Rozdzial_10/RekurencyjneLiczenieSilni.java

```
public class RekurencyjneLiczenieSilni {
  public static void main(String[] args) {
    System.out.println("Silnia 0 wynosi: " + silnia(0));
    System.out.println("Silnia 1 wynosi: " + silnia(1));
    System.out.println("Silnia 5 wynosi: " + silnia(5));
    System.out.println("Silnia 10 wynosi: " + silnia(10));
    System.out.println("Silnia 12 wynosi: " + silnia(12));
  }
```

```
  public static int silnia(int n) {
    if (n < 0) {
      return -1;
    } else if (n == 0) {
      return 1;
    } else {
      return silnia(n - 1) * n;
    }
  }
}
```

Rekurencyjna metoda `silnia` liczy silnię podanej liczby mnożąc ją przez wynik wywołania samej siebie z argumentem pomniejszonym o `1`. Na początku metody sprawdzamy, czy argument jest w ogóle poprawny – jeżeli nie, to zwrócimy liczbę `-1`. Warunek z sekcji `else if` odpowiada za przerwanie rekurencji, natomiast w sekcji `else` metoda rekurencyjnie wywołuje samą siebie.

Po uruchomieniu tego programu na ekranie zobaczymy:

```
Silnia 0 wynosi: 1
Silnia 1 wynosi: 1
Silnia 5 wynosi: 120
Silnia 10 wynosi: 3628800
Silnia 12 wynosi: 479001600
```

10.3.2.5. Liczenie kwadratu liczby na dwa sposoby

Napisz program, który będzie podnosił podaną liczbę do drugiej potęgi (liczył jej kwadrat). Program powinien mieć dwa tryby działania:

- jeżeli użytkownik przekaże do programu jeden argument, to program policzy i wypisze kwadrat tej wartości,
- jeżeli użytkownik przekaże zero argumentów, to program poprosi użytkownika o podanie liczby i wypisze jej kwadrat.

Jeżeli użytkownik przekaże więcej niż jeden argument lub przekaże jeden argument będący tekstem `--help`, to program wypisze na ekran informacje: jakie zadanie wykonuje, jakie ma tryby działania, jakiego argumentu (ewentualnie) się spodziewa, oraz co się stanie, gdy użytkownik poda argument nie będący poprawną liczbą.

Podpowiedź: aby zamienić argument programu będący stringiem na liczbę, skorzystaj z metody `Integer.parseInt`. Ta metoda spowoduje zakończenie się programu błędem, jeżeli przekazana jej wartość typu `String` nie będzie poprawną liczbą.

Zadaniem metody `main` w proponowanym rozwiązaniu będzie oddelegowanie wykonania odpowiedniej operacji w zależności od tego, czy i jakie argumenty użytkownik przekazał do programu.

W linii `(1)` sprawdzamy, czy przekazano zero argumentów – w takim przypadku, zgodnie z wymaganiami zadania, program będzie działa w trybie „interaktywnym", tzn. poprosi użytkownika o podanie liczby. Zajmie się tym metoda `obsluzTrybInteraktywny`.

Jeżeli użytkownik przekaże jeden argument, który nie będzie tekstem `--help`, to spróbujemy zamienić go na wartość typu `int` (za pomocą metody `parseInt`), a następnie wyświetlimy kwadrat otrzymanej w ten sposób liczby. To sprawdzenie wykonujemy w instrukcji `if` z linii `(2)`, natomiast obliczenie (ewentualnego) kwadratu liczby – w linii `(3)`.

Gdy liczba argumentów przekroczy jeden lub argumentem będzie tekst `--help`, to wyświetlimy informacje o programie. Odpowiedzialna jest za to metoda `wyswietlPomoc`, którą wywołujemy w linii `(4)`.

Pytania_i_zadania_rozwiazania/Rozdzial_10/KwadratLiczbyNaDwaSposoby.java

```
import java.util.Scanner;

public class KwadratLiczbyNaDwaSposoby {
  public static void main(String[] args) {
    if (args.length == 0) { // 1
      obsluzTrybInteraktywny();
    } else {
      if ( // 2
          args.length == 1 &&
          !"--help".equalsIgnoreCase(args[0])
      ) {
        System.out.println(
          podniesDoKwadratu(Integer.parseInt(args[0])) // 3
        );
      } else {
        wyswietlPomoc();
      }
    }
  }

  public static void obsluzTrybInteraktywny() {
    System.out.print(
        "Podaj liczbę a ja policzę jej kwadrat: "
    );
    int liczba = pobierzLiczbe();

    System.out.println(
```

```
            "Kwadrat tej liczby wynosi: " +
            podniesDoKwadratu(liczba)
        );
    }

    public static int pobierzLiczbe() {
        return new Scanner(System.in).nextInt();
    }

    public static int podniesDoKwadratu(int liczba) {
        return liczba * liczba;
    }

    public static void wyswietlPomoc() {
        System.out.println("""
            Program liczący kwadrat podanej liczby.

            Jeżeli program otrzyma jeden argument, to wyświetli
            jego kwadrat. Program spodziewa się argumentu
            będącego liczbą (o ile nie będzie to argument --help,
            który powoduje wyświetlenie informacji o programie).
            Nieprawidłowy argument spowoduje zakończenie
            działania programu niepowodzeniem.

            Jeżeli program nie otrzyma argumentów lub
            otrzyma więcej niż jeden argument, to
            będzie działał w trybie interaktywnym.

            Użycie:
              KwadratLiczbyNaDwaSposoby 10

            Argumenty:
              --help wyświetla informacje o programie
            """);
    }
}
```

Metoda `obsluzTrybInteraktywny` wypisuje prośbę o podanie liczby, a następnie wypisuje jej kwadrat. Używana jest w niej metoda `podniesDoKwadratu`, z której korzystamy także w metodzie `main` do obsługi trybu programu, w którym podnosi on do kwadratu liczbę otrzymaną jako argument (w linii `(3)`). Metoda `wyswietlPomoc` wypisuje informacje o programie zgodnie z wymaganiami zadania. Kilka przykładowych uruchomień tego programu:

```
> java KwadratLiczbyNaDwaSposoby
Podaj liczbę a ja policzę jej kwadrat: 5
Kwadrat tej liczby wynosi: 25
```

```
> java KwadratLiczbyNaDwaSposoby
Podaj liczbę a ja policzę jej kwadrat: abc
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)
    at java.base/java.util.Scanner.next(Scanner.java:1619)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)
    at KwadratLiczbyNaDwaSposoby.pobierzLiczbe(KwadratLiczbyNaDwaSposoby.java:33)
    at     KwadratLiczbyNaDwaSposoby.obsluzTrybInteraktywny(KwadratLiczbyNaDwaSpo-
soby.java:25)
    at KwadratLiczbyNaDwaSposoby.main(KwadratLiczbyNaDwaSposoby.java:6)
```

```
> java KwadratLiczbyNaDwaSposoby --help
Program liczący kwadrat podanej liczby.

Jeżeli program otrzyma jeden argument, to wyświetli
jego kwadrat. Program spodziewa się argumentu
będącego liczbą (o ile nie będzie to argument --help,
który powoduje wyświetlenie informacji o programie).
Nieprawidłowy argument spowoduje zakończenie
działania programu niepowodzeniem.

Jeżeli program nie otrzyma argumentów lub
otrzyma więcej niż jeden argument, to
będzie działał w trybie interaktywnym.

Użycie:
  KwadratLiczbyNaDwaSposoby 10

Argumenty:
  --help wyświetla informacje o programie
```

```
> java KwadratLiczbyNaDwaSposoby 10
100
```

10.3.2.6. Metoda znajdująca najczęściej występującą liczbę w tablicy

Napisz metodę, która przyjmie jako argument tablicę wartości typu `int` i zwróci liczbę, która w tej tablicy występuje najczęściej. Jeżeli kilka liczb występuje tyle samo (najwięcej) razy, to powinna zostać zwrócona pierwsza z nich.

Załóż, że metoda otrzymuje zawsze wartości w tablicy, które są posortowane niemalejąco. Ponadto załóż, że metoda nie otrzyma nigdy wartości `null` ani pustej tablicy. Przygotuj komentarz dokumentacyjny, w którym opiszesz działanie metody i założenia odnośnie przekazywanych do niej danych.

Przykładowe uruchomienia i spodziewane wyniki:

```
najczesciejWystepujacaLiczba(
    new int[] { 1, 1, 1, 9, 9, 10 }
) -> 1
najczesciejWystepujacaLiczba(new int[] { 5, 10, 15 }) -> 5
```

Zgodnie z wymaganiami zadania mamy założyć, że tablica, w której będziemy szukać najczęściej występującej liczby, jest posortowana niemalejąco. Dzięki temu wszystkie takie same liczby będą ustawione koło siebie, więc łatwo będziemy mogli policzyć ile razy wystąpiła każda z nich.

Istotne jest tutaj to, że nie musimy „pamiętać" ile razy wystąpiła każda liczba – wystarczy, że zapamiętamy ile razy wystąpiła liczba, którą aktualnie przetwarzamy i jeżeli wystąpiła więcej razy, niż poprzednia najczęstsza liczba, to zapiszemy te dwie informacje: jaka to liczba i ile razy wystąpiła.

Spójrzmy na proponowane rozwiązanie. W metodzie `main` wielokrotnie wywoływana jest metoda `najczesciejWystepujacaLiczba` z rożnych zestawem danych wejściowych, a wynik jej działania jest wypisywany na ekran:

Pytania_i_zadania_rozwiazania/Rozdzial_10/NajczestszaLiczbaWTablicy.java

```
public class NajczestszaLiczbaWTablicy {
  public static void main(String[] args) {
    System.out.println(
        najczesciejWystepujacaLiczba(new int[] { 1, 1, 1 })
    );
    System.out.println(
        najczesciejWystepujacaLiczba(new int[] { 1, 2, 2 })
    );
    System.out.println(
        najczesciejWystepujacaLiczba(new int[] { 1 })
    );
    System.out.println(
        najczesciejWystepujacaLiczba(new int[] { 1, 2 })
    );
    System.out.println(
        najczesciejWystepujacaLiczba(new int[] { 1, 2, 2, 2 })
    );
    System.out.println(
        najczesciejWystepujacaLiczba(
            new int[] { 1, 2, 2, 2, 3, 3, 3, 3 }
        )
    );
  }
```

```
  /**
   * Zwraca liczbę, która występuje najwięcej razy
   * w podanej tablicy, która powinna być posortowana
   * niemalejąco. Jeżeli więcej niż jedna liczba
   * występuje najwięcej razy, to zostanie zwrócona
   * pierwsza z nich. Dla nieposortowanej tablicy
   * wynik jest niezdefiniowany.
   *
   * @param liczby  niepusta i nienullowa tablica
   *                wartości posortowanych niemalejąco
   * @return        najczęściej występująca liczba
   *                w przekazanej tablicy
   */
  public static int najczesciejWystepujacaLiczba(
      int[] liczby
  ) {
    int najczesciejWystepujacaLiczba = liczby[0];
    int aktualnaLiczba = liczby[0];
    int ileRazyWystapilaNajczestszaDoTejPory = 1;
    int aktualnaIleRazyWystapila = 1;

    for (int i = 1; i < liczby.length; i++) { // 1
      if (liczby[i] == aktualnaLiczba) { // 2
        aktualnaIleRazyWystapila++;
      }
      if ( // 3
          liczby[i] != aktualnaLiczba ||
          i == liczby.length - 1
      ) {
        if ( // 4
            aktualnaIleRazyWystapila >
            ileRazyWystapilaNajczestszaDoTejPory
        ) {
          najczesciejWystepujacaLiczba =
              aktualnaLiczba; // 5

          ileRazyWystapilaNajczestszaDoTejPory =
              aktualnaIleRazyWystapila; // 6
        }

        aktualnaIleRazyWystapila = 1; // 7
        aktualnaLiczba = liczby[i]; // 8
      }
    }

    return najczesciejWystepujacaLiczba;
  }
}
```

Przed metodą `najczesciejWystepujacaLiczba` znajduje się komentarz dokumentacyjny, który opisuje co robi ta metoda i jakie założenia przyjmuje odnośnie swojego argumentu.

Na początku tej metody zdefiniowanych jest kilka zmiennych, które będę potrzebne do wyznaczenia najczęściej występującej liczby:

- `najczesciejWystepujacaLiczba` – w tej zmiennej będziemy zapisywać informację która liczba wystąpiła do tej pory najwięcej razy. Inicjalizujemy ją wartością pierwszego elementu przekazanej tablicy – nie byłoby to wymagane gdyby nie szczególny przypadek – gdy tablica będzie zawierać dokładnie jeden element, to pętla nie wykona się ani razu. Bez tej inicjalizacji kompilator zgłosi błąd, bo do tej zmiennej nie zostanie przypisana wartość.
- `aktualnaLiczba` – jest to liczba, którą porównujemy do następnej liczby z tablicy – jeżeli będą takie same, to zwiększymy licznik zawierający informację ile ta (aktualna) liczba wystąpiła w tablicy do tej pory.
- `ileRazyWystapilaNajczestszaDoTejPory` i `aktualnaIleRazyWystapila` – te dwie zmienne przechowują informację: ile razy dotychczas wystąpiła najczęstsza liczba i ile razy wystąpiła liczba, którą aktualnie przetwarzamy w tablicy.

W pętli zaczynającej się w linii `(1)` zaczynamy przetwarzanie tablicy od drugiego elementu, ponieważ pierwszy przypisaliśmy do zmiennej `aktualnaLiczba`. W instrukcji `if` z linii `(2)` sprawdzamy, czy liczba z tablicy, której indeks w tym obiegu wyznacza zmienna `i`, jest taka sama, jak ostatnio zapamiętana liczba. Jeżeli tak, to zwiększamy licznik wystąpień tej liczby.

Jeżeli liczba jest inna lub dotarliśmy do końca tablicy (linia `(3)`) to sprawdzamy, czy liczba, którą przetwarzaliśmy, wystąpiła więcej razy, niż poprzednia najczęstsza liczba (linia `(4)`). Jeżeli tak, to w linii `(5)` zapamiętujemy tę liczbę, a w linii `(6)` – ile razy wystąpiła.

Niezależnie od tego, czy ta przetwarzana liczba wystąpiła więcej razy, niż najczęstsza do tej pory, musimy przypisać do zmiennej `aktualnaLiczba` nową „aktualną" liczbę i wyzerować licznik jej wystąpień (linie `(7)` i `(8)`). Na końcu metody pozostaje nam zwrócić znalezioną najczęstszą liczbę.

Po uruchomieniu tego programu na ekranie zobaczymy:

```
1
2
1
1
2
3
```

10.3.2.7. Napisz metodę wypisującą aktualną datę

Napisz metodę, która wypisze aktualną datę w postaci:

```
Dzisiaj jest poniedziałek, 1 dzień kwietnia 2024 roku.
```

Użyj typu `LocalDate`. Aby pobrać aktualną datę, użyj metody `LocalDate.now`:

```
LocalDate aktualnaData = LocalDate.now();
```

Mając aktualną datę, możesz dowiedzieć się jaki jest aktualny dzień tygodnia (będzie to liczba z przedziału od `1` do `7`), aktualny miesiąc (liczba z przedziału od `1` do `12`), aktualny dzień miesiąca oraz rok:

```
int numerDniaTygodnia =
    aktualnaData.getDayOfWeek().getValue();
int numerMiesiaca = aktualnaData.getMonthValue();
int numerDniaMiesiaca = aktualnaData.getDayOfMonth();
int rok = aktualnaData.getYear();
```

Rozwiązanie tego zadania sprowadza się do użycia zaprezentowanych w treści zadania metod obiektu typu `LocalDate` oraz do zamiany numerów dnia miesiąca i dnia tygodnia na tekst.

W metodzie `main` w proponowanym rozwiązaniu wywoływana jest metoda, której zadaniem jest wypisanie na ekran aktualnej daty.

Na początku metody `wypiszAktualnaDate` pobieramy numery: dnia tygodnia, miesiąca, oraz dnia miesiąca, a także rok, za pomocą kilku metod wywoływanych na obiekcie typu `LocalDate`, który definiujemy w linii `(1)`. W dalszej części tej metody wypisujemy na ekran ładnie sformatowany komunikat. Do zamiany numeru dnia tygodnia i numeru dnia miesiąca na nazwy korzystamy z dwóch dedykowanych metod: `zamienDzienTygodniaNaNazwe` oraz `zamienNumerMiesiacaNaNazwe`.

Pytania_i_zadania_rozwiazania/Rozdzial_10/WypisywanieAktualnejDaty.java

```
import java.time.LocalDate;

public class WypisywanieAktualnejDaty {
  public static void main(String[] args) {
    wypiszAktulnaDate();
  }
```

```
public static void wypiszAktulnaDate() {
  LocalDate aktualnaData = LocalDate.now(); // 1
  int numerDniaTygodnia =
      aktualnaData.getDayOfWeek().getValue();
  int numerMiesiaca = aktualnaData.getMonthValue();
  int numerDniaMiesiaca = aktualnaData.getDayOfMonth();
  int rok = aktualnaData.getYear();

  System.out.printf(
      "Dzisiaj jest %s, %d dzień %s, %d roku. ",
      zamienDzienTygodniaNaNazwe(numerDniaTygodnia),
      numerDniaMiesiaca,
      zamienNumerMiesiacaNaNazwe(numerMiesiaca),
      rok
  );
}

public static String zamienDzienTygodniaNaNazwe(
    int numerDniaTygodnia
) {
  return switch (numerDniaTygodnia) {
    case 1 -> "poniedziałek";
    case 2 -> "wtorek";
    case 3 -> "środa";
    case 4 -> "czwartek";
    case 5 -> "piątek";
    case 6 -> "sobota";
    case 7 -> "niedziela";
    default -> "nieznany dzień";
  };
}

public static String zamienNumerMiesiacaNaNazwe(
    int numerMiesiaca
) {
  return switch (numerMiesiaca) {
    case 1 -> "stycznia";
    case 2 -> "lutego";
    case 3 -> "marca";
    case 4 -> "kwietnia";
    case 5 -> "maja";
    case 6 -> "czerwca";
    case 7 -> "lipca";
    case 8 -> "sierpnia";
    case 9 -> "września";
    case 10 -> "października";
```

```
            case 11 -> "listopada";
            case 12 -> "grudnia";
            default -> "nieznany miesiąc";
        };
    }
}
```

Te dwie pomocnicze metody przyjmują po jednym argumencie – numerze, odpowiednio, dnia tygodnia i miesiąca. Zwracaną z nich wartością jest wynik wyrażenia `switch`. Zwróć uwagę, że oba wyrażenia `switch` zawierają sekcję `default` – jest ona wymagana, bo w przeciwnym razie kod by się nie skompilował. Wyrażenia `switch` muszą albo brać pod uwagę wszystkie możliwe wartości podczas dopasowywania podanej wartości, albo posiadać sekcję `default`.

Przykładowe uruchomienie:

```
Dzisiaj jest piątek, 19 dzień kwietnia, 2024 roku.
```

10.3.3. Bardziej wymagające zadania

10.3.3.1. Szyfrowanie zdania

Napisz program, który będzie szyfrował bądź odszyfrowywał przekazany do niego argument. Szyfrowanie powinno działać następująco: po każdej literze powinien do szyfrowanego tekstu zostać dodany znak `'x'`. Przykład:

Niezaszyfrowany tekst											
A	l	a		m	a		k	o	t	a	
Zaszyfrowany tekst											
A	x	l	x	a	x		m	x	a	x		k	x	o	x	t	x	a	x

Tabela 1 – Przykład tekstu przed i po zaszyfrowaniu

Program powinien:

- Wyświetlać informacje o zadaniu, jakie wykonuje, oraz argumentach, jakie przyjmuje, jeżeli użytkownik nie przekaże do programu żadnego argumentu lub przekaże argument `--help`.
- Wyświetlać informacje, że podane argumenty są nieprawidłowe, jeżeli użytkownik nie poda dokładnie dwóch argumentów, gdzie pierwszy będzie jedną z następujących wartości: `-e`, `--encode`, `-d`, `--decode`.
- Szyfrować swój drugi argument, jeżeli pierwszy przekazany do programu argument to `-e` lub `--encode`.
- Odszyfrować swój drugi argument, jeżeli pierwszy przekazany do programu argument to `-d` lub `--decode`.

Zaszyfrowany lub odszyfrowany tekst powinien zostać wypisany na ekran. Wielkość liter dla opcji `--help`, `-e`, `--encode`, `-d`, oraz `--decode`, nie powinna mieć znaczenia.

Przykład uruchomienia programu do zaszyfrowania i odszyfrowania tekstu:

```
> java Szyfrator.java -e "Ala ma kota"
Axlxax mxax kxoxtxax

> java Szyfrator.java -de "Axlxax mxax kxoxtxax"
Ala ma kota
```

W metodzie `main` proponowanego rozwiązania w pierwszej kolejności sprawdzamy, czy należy wyświetlić informacje o programie (linia `(1)`). Pomocnicza metoda `czyWyswietlicPomoc` sprawdza, czy do programu przekazano zero argumentów lub czy pierwszym argumentem jest tekst `--help` (linia `(5)`). Jeżeli tak, to wywołanie metody `wyswietlPomoc` spowoduje wypisanie na ekran informacji o programie i spodziewanych przez niego argumentów.

Gdy „pomoc” programu nie zostanie wyświetlona, to sprawdzimy w linii (2) czy program otrzymał poprawne argumenty. Odpowiedzi udzieli w tym przypadku metoda `czyPoprawneArgumenty`, która sprawdza, czy przekazano dokładnie dwa argumenty i czy pierwszy z nich to jedna z wymaganych opcji (linia (6)). Dla nieprawidłowych argumentów wyświetlimy stosowną informację (linia (4)).

Jeżeli argumenty okażą się poprawne, to wypiszemy na ekran wynik działania metody `przetworzDane` (linia (3)).

Pytania_i_zadania_rozwiazania/Rozdzial_10/Szyfrator.java

```
public class Szyfrator {
  public static void main(String[] args) {
    if (czyWyswietlicPomoc(args)) { // 1
      wyswietlPomoc();
    } else {
      if (czyPoprawneArgumenty(args)) { // 2
        String trybDzialania = args[0];
        String dane = args[1];
        System.out.println(
            przetworzDane(trybDzialania, dane) // 3
        );
      } else {
        poinformujZeNieprawidloweDane(); // 4
      }
    }
  }

  public static boolean czyWyswietlicPomoc(String[] args) {
    return args.length == 0 ||
           "--help".equalsIgnoreCase(args[0]); // 5
  }

  public static void wyswietlPomoc() {
    System.out.println("""
        Program szyfruje bądź odszyfrowuje swój argument.
        Szyfr polega na dodaniu litery 'x' po każdym znaku
        nie będącym spacją.

        Użycie: Szyfrator -e "Ala ma kota"
                Szyfrator -d "Axlxax mxax kxoxtxax"

        Opcje:
         --help          wyświetla informacje o programie
         -e lub --encode  szyfruje argument następujący po tej opcji
         -d lub --decode  odszyfrowuje argument następujący po tej opcji
        """);
  }
```

```
public static boolean czyPoprawneArgumenty(String[] args) {
  return args.length == 2 && // 6
      ("--encode".equalsIgnoreCase(args[0]) ||
      "-e".equalsIgnoreCase(args[0]) ||
      "--decode".equalsIgnoreCase(args[0]) ||
      "-d".equalsIgnoreCase(args[0]));
}

public static String przetworzDane(
    String trybDzialania, String dane
) {
  if ("-e".equalsIgnoreCase(trybDzialania) ||
      "--encode".equalsIgnoreCase(trybDzialania)) {
    return zaszyfruj(dane); // 7
  } else {
    return odszyfruj(dane); // 8
  }
}

public static void poinformujZeNieprawidloweDane() {
  System.out.println("""
      Nieprawidłowe argumenty.
      Użyj argumentu --help aby wyświetlić \
      informacje o programie.
      """
  );
}

public static String zaszyfruj(String tekst) {
  String wynik = "";

  for (char znak : tekst.toCharArray()) { // 9
    wynik += Character.isLetter(znak) ? znak + "x" : znak;
  }

  return wynik;
}

public static String odszyfruj(String tekst) {
  String wynik = "";

  for (int i = 0; i < tekst.length();) {
    char znak = tekst.charAt(i);
    i += Character.isLetter(znak) ? 2 : 1;
    wynik += znak;
  }
```

```
    return wynik;
  }
}
```

Zadaniem metody `przetworzDane` jest zaszyfrowanie (linia `(7)`) lub odszyfrowanie (linia `(8)`) przekazanego do niej tekstu w zależności od wartości jej pierwszego argumentu.

Szyfrowanie odbywa się w metodzie `zaszyfruj`. Zgodnie z wymaganiami zadania mamy wstawić literę ‘x’ po każdej literze w szyfrowanym tekście. Aby sprawdzić, czy dany znak jest literą, możemy użyć metody `Characater.isLetter`. W metodzie `szyfruj` przechodzimy przez wszystkie znaki w tekście korzystając z pętli `for`-each, która przejdzie wszystkie elementy tablicy zwróconej przez metodę `toCharArray` (linia `(9)`). Ta metoda zwraca tablicę wartości typu `char` – kolejne elementy to znaki w stringu, na rzecz którego ta metoda została wywołana. W ciele pętli używamy metody `Character.isLetter` aby sprawdzić, czy mamy do wynikowego (zaszyfrowanego) tekstu dodać literę ‘x’, czy nie.

Metoda `zaszyfruj` działa nieco inaczej – zmiany wartości zmiennej pętli `i` musimy uzależnić od tego, czy aktualnie przetwarzany znak jest literą, czy nie. Jeżeli tak, to znaczy, że po tej literze znajduje się litera ‘x’, dodana tam w procesie szyfrowania – w takim przypadku musimy „przeskoczyć” tę literę, więc zwiększymy wartość zmiennej `i` o `2`:

```
i += Character.isLetter(znak) ? 2 : 1;
```

Do sprawdzenia, czy aktualny znak jest literą, ponownie posłuży nam metoda `Character.isLetter`.

Kilka przykładowych uruchomień tego programu:

```
> java Szyfrator
Program szyfruje bądź odszyfrowuje swój argument.
Szyfr polega na dodaniu litery 'x' po każdym znaku
nie będącym spacją.

Użycie: Szyfrator -e "Ala ma kota"
        Szyfrator -d "Axlxax mxax kxoxtxax"

Opcje:
  --help            wyświetla informacje o programie
  -e lub --encode   szyfruje argument następujący po tej opcji
  -d lub --decode   odszyfrowuje argument następujący po tej opcji
```

```
> java Szyfrator -e
Nieprawidłowe argumenty.
Użyj argumentu --help aby wyświetlić
informacje o programie.
```

```
> java Szyfrator --encode "Ala ma kota."
Axlxax mxax kxoxtxax.
```

```
> java Szyfrator -d "Axlxax mxax kxoxtxax."
Ala ma kota.
```

10.3.3.2. Zliczanie wystąpień stringa

Napisz program, który przyjmie dwa argumenty i odpowie na pytanie, ile razy drugi argument występuje w pierwszym. Dla przykładu, jeżeli program zostałby wywołany z argumentami `"Ala ma kota, kot ma sielskie życie."` oraz `"ma"`, to program wypisałby wynik `2`, ponieważ w podanym zdaniu ciąg znaków "ma" występuje dwa razy. Uwaga: małe i wielkie litery powinny być uznawane za różne.

Program powinien wypisywać informacje o tym, jakie zadanie wykonuje i jakie argumenty przyjmuje, jeżeli: użytkownik uruchomi go bez argumentów, z pierwszym argumentem będącym tekstem `--help`, lub z więcej niż dwoma argumentami.

Podpowiedź: skorzystaj z metody `indexOf` typu `String`, z jej przeładowanej wersji, która przyjmuje dwa argumenty. Jak działa ta metoda dowiesz się z oficjalnej dokumentacji Biblioteki Standardowej Java:

https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/String.html#indexOf(java.lang.String,int)

Przykładowe uruchomienie programu:

```
> java LiczenieWystapienStringa.java "Ala ma kota" a
3

> java LiczenieWystapienStringa.java "Ala ma kota, kot ma sielskie życie." ma
2
```

Jedna z wersji metody `indexOf` (znajdującej się w typie `String`) przyjmuje dwa argumenty – pierwszy to łańcuch tekstowy, który chcemy znaleźć w stringu, na rzecz którego ta metoda jest wywoływana, a drugi to indeks w tym stringu od którego poszukiwanie powinno się zacząć. Jeżeli wskazany fragment tekstu nie zostanie znaleziony, to metoda `indexOf` zwróci wartość `-1`.

Skorzystamy z tej metody w następujący sposób: w pętli będziemy szukać podanego przez użytkownika tekstu do czasu, aż metoda `indexOf` nie zwróci liczby `-1`. Dodatkowo, za każdym razem, gdy w pętli wywołamy metodę `indexOf`, to jako drugi argument przekażemy zwrócony przez tę metodę wcześniej indeks, gdzie tekst został znaleziony, powiększony o długość szukanego tekstu. W ten sposób metoda `indexOf` będzie znajdowała (ewentualne) kolejne wystąpienia szukanego tekstu w źródłowym stringu.

Na początku metody `main` w proponowanym rozwiązaniu sprawdzamy, czy wyświetlić pomoc. Jeżeli informacje o programie nie będę wyświetlone, to wypiszemy wynik wywołania metody `policzWystapienia`. Jako jej argumenty podajemy pierwszy i drugi argument przekazany do naszego programu – są to, odpowiednio, tekst, w którym szukamy pewnego fragmentu, i ten fragment (linia `(1)`).

Pytania_i_zadania_rozwiazania/Rozdzial_10/LiczenieWystapienStringa.java

```
public class LiczenieWystapienStringa {
  public static void main(String[] args) {
    if (czyWyswietlicPomoc(args)) {
      wyswietlPomoc();
    } else {
      System.out.println(
          policzWystapienia(args[0], args[1]) // 1
      );
    }
  }

  public static boolean czyWyswietlicPomoc(String[] args) {
    return args.length == 0 ||
        "--help".equalsIgnoreCase(args[0]) ||
        args.length != 2;
  }

  public static void wyswietlPomoc() {
    System.out.println("""
        Program zlicza ile razy jego drugi argument
        występuje w pierwszym argumencie. Wielkość
        liter jest rozróżniana.

        Użycie: LiczenieWystapienStringa "Ala ma kota" ma

        Opcje:
          --help  wyświetla informacje o programie
        """);
  }
```

```
  public static int policzWystapienia(
      String tekst, String szukany
  ) {
    int liczbaWystapien = 0;

    if (tekst != null && szukany != null) {
      int pozycjaWystapienia = tekst.indexOf(szukany); // 2

      while (pozycjaWystapienia != -1) { // 3
        pozycjaWystapienia = tekst.indexOf(
            szukany, pozycjaWystapienia + szukany.length()
        );
        liczbaWystapien++;
      }
    }

    return liczbaWystapien;
  }
}
```

W metodzie `policzWystapienia` w linii `(2)` po raz pierwszy szukamy fragmentu tekstu w przekazanym argumencie. Jednorazowo skorzystamy z wersji metody `indexOf`, która przyjmuje tylko jeden argument – chcemy wyszukać pierwsze (ewentualne) wystąpienia danego stringa. W pętli zaczynającej się w linii `(3)` sprawdzamy wynik wywołania metody `indexOf` – jeżeli zwróciła `-1`, to znaczy, że szukany fragment nie został znaleziony. W przeciwnym razie ponownie korzystamy z metody `indexOf`, ale tym razem wskazujemy indeks od którego ta metoda ma zacząć szukanie tekstu – jest to indeks poprzedniego wystąpienia znalezionego stringa powiększony o długość tego stringa. Dzięki temu kolejne wywołana metody `indexOf` będą szukały tekstu „coraz dalej” w źródłowym stringu. W pętli zwiększamy także licznik wystąpień szukanego tekstu. Kilka przykładowych uruchomień tego programu:

```
> java LiczenieWystapienStringa --help
Program zlicza ile razy jego drugi argument
występuje w pierwszym argumencie. Wielkość
liter jest rozróżniana.

Użycie: LiczenieWystapienStringa "Ala ma kota" ma

Opcje:
  --help  wyświetla informacje o programie
```

```
> java LiczenieWystapienStringa "Ala ma kota" a
3
```

```
> java LiczenieWystapienStringa "Ala ma kota" A
1
```

10.3.3.3. Lista zadań

Napisz program, który będzie przechowywał listę zadań i pozwalał na usuwanie z niej tych, które zostały wykonane. Gdy lista zadań się wyczerpie, program zakończy działanie. Program po uruchomieniu powinien zapytać użytkownika ile zadań chce podać, a następnie powinien po kolei odczytać każde z zadań. Aby pobrać całe zdanie od użytkownika, skorzystaj z poniższej metody:

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

Po pobraniu zadań program powinien je wyświetlać w pętli i pytać, które zadanie użytkownik wykonał, odczytując numer zadania, które ma zostać usunięte. Przykładowe uruchomienie tego programu może wyglądać następująco:

```
Ile zadań masz do wykonania? 3
Podaj zadanie nr #1: Pobawić się z kotem.
Podaj zadanie nr #2: Nakarmić kota.
Podaj zadanie nr #3: Zamówić jedzenie dla kota.

Twoja lista zadań to:
Zadanie #1: Pobawić się z kotem.
Zadanie #2: Nakarmić kota.
Zadanie #3: Zamówić jedzenie dla kota.

Które zadanie wykonałeś(-aś)? 2
Super! Jedno zadanie z głowy!
Twoja lista zadań to:
Zadanie #1: Pobawić się z kotem.
Zadanie #2: Zamówić jedzenie dla kota.

Które zadanie wykonałeś(-aś)? 0
Nieprawidłowy numer zadania.

Twoja lista zadań to:
Zadanie #1: Pobawić się z kotem.
Zadanie #2: Zamówić jedzenie dla kota.

Które zadanie wykonałeś(-aś)? 1
Super! Jedno zadanie z głowy!

Twoja lista zadań to:
Zadanie #1: Zamówić jedzenie dla kota.

Które zadanie wykonałeś(-aś)? 1
Super! Jedno zadanie z głowy!

Brak zadań - siesta.
```

Z racji tego, że użytkownik będzie mógł wskazać, że wykonał pewne zadanie, będziemy musieli mieć możliwość usuwania elementu z tablicy, w której będziemy przechowywać listę zadań. To zadanie oddelegujemy do metody `usunElement`, która będzie przyjmować tablicę oraz indeks elementu, który ma zostać z niej usunięty.

Rozmiaru tablic nie można zmieniać w Javie, więc utworzymy nową tablicę o rozmiarze o jeden mniejszym, niż tablica źródłowa, i zwrócimy ją z metody `usunElement`. Ta nowa tablica będzie kopią tablicy przekazanej jako argument z wyjątkiem jednego elementu – tego, który ma zostać usunięty – będzie on pominięty podczas przepisywania elementów z oryginalnej tablicy do kopii. Implementacja tej metody zaczyna się w linii `(9)`.

W rozwiązaniu tego zadania skorzystamy jeszcze z dwóch innych pomocniczych metod (nie licząc metod wczytujących dane od użytkownika). Pierwsza z nich to `wczytajZadania` – ta metoda w pętli pobierze od użytkownika listę zadań i zwróci je w postaci tablicy wartości typu `String`. Ta metoda zaczyna się w linii `(7)`. Druga pomocnicza metoda to `wyswietlZadania` – zgodnie z nazwą, jej zadaniem będzie wypisanie na ekran zadań, które zostały jeszcze do wykonania (linia `(8)`).

Te metody używane są w metodzie `main` w proponowanym rozwiązaniu. W linii `(1)` wczytujemy od użytkownika listę zadań. Dopóki lista zadań nie jest pusta (warunek z linii `(2)`), będziemy wykonywać następujące operacje:

- linia `(3)` – wyświetlamy zadania, które znajdują się obecnie na liście zadań,
- linia `(4)` – pytamy użytkownika które zadanie wykonał,
- linia `(5)` – sprawdzamy, czy podany został poprawny numer zadania,
- linia `(6)` – jeżeli tak, to usuwamy to wykonane zadanie z listy zadań.

Pytania_i_zadania_rozwiazania/Rozdzial_10/ListaZadan.java

```
import java.util.Scanner;

public class ListaZadan {
  public static void main(String[] args) {
    String[] zadania = wczytajZadania(); // 1

    while (zadania.length > 0) { // 2
      wyswietlZadania(zadania); // 3

      // 4
      System.out.print("\nKtóre zadanie wykonałeś(-aś)? ");
      int wykonaneZadanie = pobierzLiczbe() - 1;

      if ( // 5
          wykonaneZadanie < 0 ||
          wykonaneZadanie >= zadania.length
      ) {
        System.out.println("Nieprawidłowy numer zadania.");
```

```
      } else { // 6
        System.out.println("Super! Jedno zadanie z głowy!");
        zadania = usunElement(zadania, wykonaneZadanie);
      }
    }

    System.out.println("\nBrak zadań - siesta.");
  }

  public static String[] wczytajZadania() { // 7
    System.out.print("Ile zadań masz do wykonania? ");
    int ileZadan = pobierzLiczbe();

    String[] zadania = new String[Math.max(ileZadan, 0)];

    for (int i = 0; i < ileZadan; i++) {
      System.out.printf("Podaj zadanie nr #%d: ", i + 1);
      zadania[i] = pobierzLinieTekstu();
    }

    return zadania;
  }

  // 8
  public static void wyswietlZadania(String[] zadania) {
    System.out.println("\nTwoja lista zadań to:");

    for (int i = 0; i < zadania.length; i++) {
      System.out.printf(
          "Zadanie #%d: %s\n", i + 1, zadania[i]
      );
    }
  }

  public static String[] usunElement( // 9
      String[] tab, int indeksElementuDoUsuniecia
  ) {
    String[] kopiaBezJednegoElementu =
        new String[tab.length - 1];

    for (int i = 0; i < tab.length; i++) {
      if (i < indeksElementuDoUsuniecia) {
        kopiaBezJednegoElementu[i] = tab[i];
      } else if (i > indeksElementuDoUsuniecia) {
        kopiaBezJednegoElementu[i - 1] = tab[i];
      }
```

```
    }

    return kopiaBezJednegoElementu;
  }

  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }

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

Działanie programu kończy się, gdy na liście zadań nie będzie już żadnych zadań. Wypiszemy wtedy jeszcze na końcu metody `main` komunikat dla użytkownika.

Przykładowy wynik uruchomienia tego programu wygląda następująco:

```
> java ListaZadan
Ile zadań masz do wykonania? 2
Podaj zadanie nr #1: "Zrobic obiad."
Podaj zadanie nr #2: "Zjesc obiad."

Twoja lista zadań to:
Zadanie #1: "Zrobic obiad."
Zadanie #2: "Zjesc obiad."

Które zadanie wykonałeś(-aś)? 2
Super! Jedno zadanie z głowy!

Twoja lista zadań to:
Zadanie #1: "Zrobic obiad."

Które zadanie wykonałeś(-aś)? 1
Super! Jedno zadanie z głowy!

Brak zadań - siesta.
```

10.3.3.4. Gra w zgadywanie hasła

Napisz program, w którym zadaniem użytkownika będzie odgadnąć losowe hasło. Hasło zdefiniuj w programie jako tablica wartości typu `String`. Zakładamy, że każde hasło to pojedyncze słowo bez polskich znaków.

Program powinien wypisywać hasło, w którym litery zastąpione są znakami podkreślenia, jeżeli litera na danej pozycji nie została jeszcze odgadnięta. Użytkownik

powinien podawać po jednej literze na raz, po czym program będzie ponownie wypisywać na ekran hasło z odkrytymi literami, które użytkownik wskazał. Jeżeli użytkownik poda więcej, niż jeden znak na raz, będzie to świadczyło o tym, że zgaduje hasło. Przykładowe uruchomienie programu:

```
Odgadnij hasło!
_ _ _ _ _ _ _ _ _ _ _ _ _

Podaj literę lub odgadnięte hasło: p
p _ _ _ _ _ _ _ _ _ _ _ _

Podaj literę lub odgadnięte hasło: a
p _ _ _ _ a _ _ _ a _ _ _

Podaj literę lub odgadnięte hasło: r
p r _ _ r a _ _ _ a _ _ _

Podaj literę lub odgadnięte hasło: g
p r _ g r a _ _ _ a _ _ _
Podaj literę lub odgadnięte hasło: programista
Nieprawidłowe hasło.

Podaj literę lub odgadnięte hasło: w
p r _ g r a _ _ w a _ _ _

Podaj literę lub odgadnięte hasło: o
p r o g r a _ o w a _ _ _

Podaj literę lub odgadnięte hasło: programowanie
Gratulacje! Odgadłeś(-aś) hasło.
```

W tym programie musimy w jakiś sposób zaznaczać litery, które udało się już odgadnąć użytkownikowi. Aby osiągnąć ten cel posłużyłem się tablicą wartości typu `char`, którą wstępnie inicjalizuję znakiem podkreślenia. Gdy użytkownik poda literę do odkrycia, to w tej tablicy, na odpowiednich miejscach, wpisana zostanie ta litera, jeżeli znajduje się ona w wylosowanym haśle.

Dla przykładu, jeżeli wylosowanym hasłem jest „hipopotam”, to tablica `odgadnieteLitery` na początku programu wygląda następująco:

Indeks	0	1	2	3	4	5	6	7	8
Znak w haśle	h	i	p	o	p	o	t	a	m
Tablica `odgadnieteLitery`	_	_	_	_	_	_	_	_	_

Tabela 2 – Zawartość tablicy odgadnieteLitery na początku działania gry w zgadywanie hasła

Gdy użytkownika wskaże, że chce odkryć literę ‘p’ (potencjalnie występującą) w haśle, to do tablicy `odgadnieteLitery` wstawiona zostanie ta litera pod indeksami `2` i `4`, ponieważ w wylosowanym haśle ta litera znajduje się dwukrotnie, właśnie pod tymi indeksami:

Indeks	0	1	2	3	4	5	6	7	8
Znak w haśle	h	i	p	o	p	o	t	a	m
Tablica `odgadnieteLitery`	_	_	p	_	p	_	_	_	_

Tabela 3 – Zawartość tablicy odgadnieteLitery po tym, jak użytkownik wskaże literę 'p' do odkrycia

Dzięki takiej reprezentacji odgadywanego hasła łatwo będzie je wyświetlać użytkownikowi – wystarczy, że wypiszemy na ekran zawartość tablicy `odgadnieteLitery`. Po każdej literze będzie dodatkowo wypisana spacja, aby oddzielić od siebie poszczególne znaki, co da lepszy efekt na ekranie.

Gra będzie działać do czasu, aż:

- użytkownik odkryje wszystkie litery,
- użytkownik odgadnie hasło, wpisując je w oknie linii poleceń.

Wynika z tego, że nasz program będzie rozróżniał dwa przypadki: podawanie litery do odkrycia oraz próbę odgadnięcia hasła. Łatwo odróżnić od siebie te dwie operacje – w przypadku pierwszej sprawdzimy, czy użytkownik podał dokładnie jeden znak – wtedy odkryjemy go w odgadywanym haśle (o ile w nim występuje), a gdy użytkownik wpisze więcej, niż jedną literę, to sprawdzimy, czy jest to zgadywane hasło.

Spójrzmy na proponowane rozwiązanie. Na początku metody `main`, w linii `(1)`, losowane jest hasło – jest to zadanie metody `wylosujHaslo`, w której zdefiniowana jest tablica wartości typu `String`, które stanowią hasła, które użytkownik może zgadywać. Ta metoda zaczyna się w linii `(12)` i korzysta z metody `nextInt` do wylosowania indeksu z tablicy hasłami – w ten sposób zostanie losowo wybrane jedno z przygotowanych haseł.

Po wylosowaniu hasła definiujemy tablicę `odgadnieteLitery`, w której, zgodnie z wcześniejszym opisem, będziemy przechowywać odgadnięte litery. Na początku żadna litera nie jest jeszcze odgadnięta przez użytkownika, więc tę tablicę inicjalizujemy znakiem podkreślenia w linii `(2)`. Zmienna `ileLiterOdgadnieto` posłuży nam do ustalenia, czy użytkownik zgadł już wszystkie litery w haśle.

Przed pętlą, w linii `(3)`, wypisujemy po raz pierwszy hasło – na razie będą to jedynie znaki podkreślenia, których liczba będzie świadczyć o liczbie znaków, z których składa się wylosowane hasło. Jeżeli wylosowanym hasłem będzie „hipopotam”, to w tym momencie użytkownik zobaczy na ekranie:

```
Odgadnij hasło!
_ _ _ _ _ _ _ _ _
```

Za wypisywanie odgadniętych liter odpowiedzialna jest metoda `wypiszHaslo`, której definicja zaczyna się w linii `(13)`. Ta prosta metoda wypisuje elementy przekazanej do niej tablicy, oddzielając je od siebie pojedynczym znakiem spacji.

W pętli `do…while`, która zaczyna się w linii `(4)`, prosimy użytkownika o podanie litery bądź o odgadnięcie hasła. Na wartości zwróconej z metody `pobierzSlowo` od razu wywołujemy metodę `toLowerCase`, dzięki czemu nie będzie miało znaczenia, czy użytkownik wpisze literę/słowo małymi czy wielkimi literami. W przeciwnym razie musielibyśmy wziąć taki przypadek pod uwagę podczas porównywania hasła i liter w haśle do litery/słowa pobranego od użytkownika. Jeżeli użytkownik poda pojedynczy znak (warunek z linii `(5)`), to przechodzimy w pętli przez wszystkie znaki w haśle i jeżeli wskazana przez użytkownika litera się w nim znajduje, to na odpowiedniej pozycji w tablicy `odgadnieteLitery` wstawimy tę literę oraz zwiększymy liczbę odgadniętych liter. Zwróć uwagę, że te operacje wykonujemy tylko wtedy, gdy dana litera nie była wcześniej odgadnięta – odpowiada za to fragment warunku w linii `(6)`. Ma to na celu obsłużenie sytuacji, w której użytkownik poda tę samą literę więcej niż raz – w takim przypadku nie chcemy ponownie zwiększać wartości zmiennej `ileLiterOdgadnieta`. Na końcu tej sekcji instrukcji `if` wypisujemy na ekran hasło z odkrytymi do tej pory literami (linia `(7)`).

Gdy użytkownik wpisze w oknie linii poleceń więcej, niż jeden znak, to będzie to oznaczało, że próbuje odgadnąć hasło – tę sytuację obsługuje sekcja `else` z linii `(8)`. Jeżeli użytkownik zgadnie hasło, to wypiszemy stosowny komunikat i przerwiemy pętlę – następuje koniec gry. Jeżeli hasło będzie nieprawidłowe, to także poinformujemy o tym fakcie użytkownika (linia `(9)`).

Pętla będzie działać do czasu, aż nie zostanie przerwana, gdy użytkownik odgadnie hasło lub wskaże wszystkie litery w haśle – ten drugi warunek znajduje się w warunku pętli `do…while` w linii `(10)`.

Na końcu programu musimy jeszcze wziąć pod uwagę, że pętla mogła się zakończyć właśnie z powodu znalezienia wszystkich liter w haśle – taka sytuacja wystąpi, gdy wartość zmiennej `ileLiterOdgadnieto` będzie równa długości wylosowanego hasła. Ten przypadek obsłużymy w linii `(11)` i jeżeli warunek będzie spełniony, to wyświetlimy odpowiedni komunikat.

Pytania_i_zadania_rozwiazania/Rozdzial_10/ZgadywanieSlowa.java

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

public class ZgadywanieSlowa {
  public static void main(String[] args) {
    String haslo = wylosujHaslo(); // 1
    char[] odgadnieteLitery = new char[haslo.length()];
    Arrays.fill(odgadnieteLitery, '_'); // 2
    int ileLiterOdgadnieto = 0;
```

```
    System.out.println("Odgadnij hasło!");
    wypiszHaslo(odgadnieteLitery); // 3

    do { // 4
      System.out.print(
          "\nPodaj literę lub odgadnięte hasło: "
      );
      String literaLubHaslo = pobierzSlowo().toLowerCase();

      if (literaLubHaslo.length() == 1) { // 5
        for (int i = 0; i < haslo.length(); i++) {
          if (
              haslo.charAt(i) == literaLubHaslo.charAt(0) &&
              odgadnieteLitery[i] == '_' // 6
          ) {
            ileLiterOdgadnieto++;
            odgadnieteLitery[i] = haslo.charAt(i);
          }
        }
        wypiszHaslo(odgadnieteLitery); // 7
      } else { // 8
        if (haslo.equals(literaLubHaslo)) {
          System.out.println(
              "Gratulacje! Odgadłeś(-aś) hasło."
          );
          break;
        } else {
          System.out.println("Nieprawidłowe hasło."); // 9
        }
      }
    } while (ileLiterOdgadnieto < haslo.length()); // 10

    if (ileLiterOdgadnieto == haslo.length()) { // 11
      System.out.println(
          "Gratulacje! Podałeś(-aś) wszystkie litery hasła."
      );
    }
  }

  public static String wylosujHaslo() { // 12
    String[] hasla = {
        "programowanie", "nauka", "hipopotam", "kapibara",
        "herbata", "java", "samolot", "lemoniada", "pizza",
        "spacer", "muzyka", "telewizor", "kosmos", "gitara"
    };
```

```
    int losowyIndeksHasla =
        ThreadLocalRandom.current().nextInt(hasla.length);

    return hasla[losowyIndeksHasla];
  }

  // 13
  public static void wypiszHaslo(char[] literyHasla) {
    for (char znak : literyHasla) {
      System.out.print(znak + " ");
    }
    System.out.println();
  }

  public static String pobierzSlowo() {
    return new Scanner(System.in).next();
  }
}
```

Dwa przykładowe uruchomienia tego programu:

```
> java ZgadywanieSlowa
Odgadnij hasło!
_ _ _ _ _

Podaj literę lub odgadnięte hasło: n
n _ _ _ _

Podaj literę lub odgadnięte hasło: a
n a _ _ a

Podaj literę lub odgadnięte hasło: u
n a u _ a

Podaj literę lub odgadnięte hasło: k
n a u k a
Gratulacje! Podałeś(-aś) wszystkie litery hasła.
```

```
> java ZgadywanieSlowa
Odgadnij hasło!
_ _ _ _ _ _ _ _ _

Podaj literę lub odgadnięte hasło: a
_ a _ _ _ a _ a

Podaj literę lub odgadnięte hasło: k
k a _ _ _ a _ a

Podaj literę lub odgadnięte hasło: b
k a _ _ b a _ a

Podaj literę lub odgadnięte hasło: kapibara
Gratulacje! Odgadłeś(-aś) hasło.
```

10.3.3.5. Gra dla dwóch osób w kółko i krzyżyk

Napisz grę w kółko i krzyżyk dla dwóch osób. Podczas ruchu danego gracza powinien on podać liczbę od `1` do `9`, która będzie symbolizować miejsce na planszy, gdzie ma zostać umieszczony znak tego gracza:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
```

Możesz łatwo wyświetlić planszę, jeżeli skorzystasz z poniższej metody:

```
public static void wyswietlPlansze(Character[] znaki) {
  System.out.printf("""
      \n %s | %s | %s
      -----------
       %s | %s | %s
      -----------
       %s | %s | %s
      \n""", znaki);
}
```

Znaki wstawiane na planszę przez graczy przetrzymuj w tablicy wartości typu `Character`, która będzie wypełniona na początku znakami spacji:

```
Character[] znaki = new Character[9];
Arrays.fill(znaki, ' ');
```

Gdy gracz wybierze miejsce, gdzie chce wstawić swój znak, powinien on zostać umieszczony w odpowiednim miejscu w zaprezentowanej tablicy. Program powinien sprawdzać, czy któryś z graczy wygrał lub czy nastąpił remis.

Starałem się proponowane rozwiązanie przygotować w taki sposób, by kod był jak najprostszy. Ogólnie najbardziej skomplikowaną częścią implementacji konsolowej wersji kółka i krzyżyk jest, moim zdaniem, sprawdzanie, czy jeden z graczy wygrał – omówienie metody, która odpowiada na to pytanie, pozostawię na koniec.

W opisie zadania przedstawiona jest instrukcja, która w wygodny sposób wypisuje na ekran linii poleceń planszę do gry w kółko i krzyżyk:

```
System.out.printf("""
    \n %s | %s | %s
    -----------
    %s | %s | %s
    -----------
    %s | %s | %s
    \n""", znaki);
```

Aby ta metoda zadziałała poprawnie dla tablicy elementów pewnego typu, ten typ musi w być typem złożonym – ta metody `printf` spodziewa się tablicy wartości typu `Object[]`. Dlatego w opisie zadania napisałem, aby tablicę ze znakami wstawionymi na planszę przez graczy zdefiniować jako tablicę wartości typu `Character`, a nie `char`. W rozdziale o typie `String` i typach złożonych w *„Javie od podstaw”* wspomniałem, że Java zawiera typy złożone, które odwzorowują typy prymitywne – typ `Character` jest właśnie takim typem. Tablic typów prymitywnych nie można rzutować na tablice typów złożonych, więc próba przekazania tablicy wartości typu `char` do używanej w tym programie wersji metody `printf` nie zadziałałaby poprawnie. Są to bardziej zaawansowane na ten moment zagadnienia, ale z tablicy wartości typu `Character` możemy w ramach rozwiązania tego zadania korzystać tak samo, jakby była to tablica wartości typu `char`. Wspominam tutaj o tym dlatego, że mogłeś(-aś) się zastanawiać dlaczego tablica `znaki`:

```
Character[] znaki = new Character[9];
Arrays.fill(znaki, ' ');
```

przechowuje wartości typu `Character`, więc chciałem to wyjaśnić.

Wywołanie metody `wypiszPlansze` na początku programu skutkuje wyświetleniem w oknie linii poleceń pustej planszy do gry w kółko do krzyżyk:

```
   |   |
-----------
   |   |
-----------
   |   |
```

Gdy któryś z graczy wykona ruch, np. pierwszy gracz wstawi swój znak ‘x’ do środka (podając liczbę `5` jako numer pola, gdzie chce wstawić znak), to plansza (po aktualizacji tablicy `znaki`) zostanie wyświetlona w następujący sposób:

```
   |   |
-----------
   | x |
-----------
   |   |
```

Po kolejnych ruchach plansza będzie się zapełniać kolejnymi znakami wstawianymi przez obu graczy. Wstawianie znaku odbywa się poprzez pobranie od użytkownika numeru pola, gdzie chce umieścić znak, a następnie wstawieniu znaku aktualnego gracza w odpowiednie miejsce tablicy `znaki`. Numery pól są następujące:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
```

Wstawienie znaku ‘x’ pierwszego gracza do piątego pola wiąże się z przypisaniem piątemu elementowi tablicy `znaki` znaku `'x'`:

```
// dla przypomnienia: indeks piątego elementu to 4,
// a nie 5, bo indeksy elementów zaczynamy liczyć od 0
znaki[4] = 'x';
```

Spójrzmy teraz na proponowane rozwiązanie tego zadania. Na początku metody `main` definiujemy tablicę, która będzie reprezentować znaki graczy umieszczane na planszy. Na początku plansza powinna być pusta, więc wypełniamy tę tablicę spacjami (linia `(1)`). W zmiennej `liczbaMozliwychRuchow` zapisujemy aktualną długość tablicy ze znakami graczy (linia `(2)`) – posłuży nam ona do sprawdzenia, czy nastąpił remis – gdy liczba możliwych ruchów dojdzie do zera, a nikt nie wygrał w ostatnim ruchu, to gra zakończy się remisem. Po każdym poprawnym ruchu gracza zmniejszymy wartość tej zmiennej o jeden.

Zmienna `znakAktualnegoGracza`, definiowana w linii `(3)`, będzie zawierała znak gracza, który ma wykonać teraz ruch. Będziemy zmieniać jej wartość po poprawnym ruchu aktualnego gracza.

Zmienna `wynik` z linii `(4)` posłuży nam jako warunek działania pętli `while`, w której będzie się wykonywać główna część gry. Gdy któryś z graczy wygra lub nastąpi remis, to przypiszemy tej zmiennej odpowiedni komunikat. Do czasu, aż się to wydarzy, ta zmienna będzie miała wartość `null`, którą zostanie zainicjalizowana w linii `(4)`. Prawdziwość warunku `wynik == null` z linii `(5)` będzie oznaczać, że gra jeszcze się nie zakończyła.

Na początku pętli `while` wyświetlamy aktualny stan planszy do gry (linia `(6)`), po czym prosimy użytkownika o podanie numeru pola, w którym chce umieścić swój znak. Odpowiedzialna jest za to metoda `pobierzRuchGracza`. Oczekuje ona jako argumentu znaku aktualnego gracza, gdyż wyświetla go w komunikacie na ekranie:

```
public static int pobierzRuchGracza(char znakGracza) {
  System.out.println("Ruch gracza '" + znakGracza + "'");
  System.out.print("\nGdzie chcesz umieścić znak? [1-9] ");
  return pobierzLiczbe() - 1;
}
```

Wartość pobrana od użytkownika jest zmniejszana o `1`, ponieważ użytkownicy będą się spodziewać, że pola na planszy są numerowane od `1`, a chcemy z tej metody otrzymać (potencjalnie) poprawny indeks w tablicy ze znakami, w której elementy (jak w każdej tablicy w języku Java) zaczynamy numerować od `0`.

W dalszej części pętli `while` sprawdzamy, czy gracz wskazał poprawne pole, w którym chce umieścić swój znak (linia `(7)`). Na to pytanie odpowiada metoda `czyRuchPoprawny`:

```
public static boolean czyRuchPoprawny(
    int ruchGracza, Character[] znaki
) {
  if (ruchGracza < 0 || ruchGracza >= znaki.length) {
    System.out.println("\n----- Nieprawidłowy ruch.\n");
    return false;
  } else if (znaki[ruchGracza] != ' ') {
    System.out.println("\n----- To pole jest już zajęte.\n");
    return false;
  }
  return true;
}
```

Jeżeli gracz podał liczbę spoza zakresu dozwolonych pól lub w polu, które wskazał, znajduje się już znak, to wyświetlimy stosowny komunikat i zwrócimy `false`. Aby sprawdzić, czy pole jest zajęte, sprawdzamy, czy wartość danego elementu w tablicy znaki jest różna niż spacja. Jeżeli ten warunek będzie spełniony, to znaczy, że któryś z graczy wstawił tam już swój znak (‘x’ lub ‘o’).

Gdy ruch gracza jest poprawny, metoda zwróci po prostu wartość `true`. W tym przypadku, w linii `(8)`, wstawimy w odpowiednie miejsce naszej „planszy" (czyli tablicy `znaki`) znak aktualnego gracza, a także zmniejszymy liczbę możliwych do wykonania ruchów (linia `(9)`).

Pozostaje nam jeszcze sprawdzić, czy gracz, który wykonał właśnie ruch, wygrał grę (linia `(10)`). Metodę, która odpowiada na to pytanie, omówię za chwilę. Jeżeli nikt nie wygrał, to sprawdzamy, czy nastąpił remis (linia `(11)`). Jeśli ktoś wygrał lub jest remis, to do zmiennej `wynik` przypiszemy komunikat informujący graczy co się wydarzyło. Spowoduje to, że będzie to ostatni obieg pętli `while`, bo warunek `wynik == null` z linii `(5)` przestanie być spełniony. Gdy pętla się zakończy, wypiszemy treść komunikatu (linia `(14)`), a także ponownie wyświetlimy planszę, aby pokazać ostatni ruch wykonany w tej grze (linia `(13)`).

Jeżeli nikt nie wygrał i nie nastąpił remis, to w linii `(12)` zmienimy znak aktualnego gracza – jeżeli w tej turze był ruch gracza wstawiającego na planszę znak 'x', to zmienimy aktualny znak na 'o', a w przeciwnym razie – na 'x'. W tym celu korzystamy z trój-argumentowego operatora warunkowego:

```
znakAktualnegoGracza =
    znakAktualnegoGracza == 'x' ? 'o' : 'x'; // 12
```

Pozostała nam jeszcze do omówienia metoda `sprawdzCzyWygrana` która sprawdza, czy ostatni ruch spowodował, że aktualny gracz wygrał. Jak możemy to zrobić? Musimy sprawdzić, czy wszystkie znaki w pierwszym rzędzie, drugim, lub trzecim są takie same, jak ostatnio wstawiony znak. Jeżeli nie, to musimy następnie sprawdzić wszystkie kolumny. Jeśli nadal nie znajdziemy trzech takich samych znaków, to pozostają nam jeszcze obie przekątne. Spójrz na poniższą reprezentację planszy do gry:

```
 1 | 2 | 3
-----------
 4 | 5 | 6
-----------
 7 | 8 | 9
```

Metoda sprawdzająca, czy wygrał jeden z graczy, powinna porównać znaki wstawione w polach:

- 1, 2, 3 – pierwszy rząd,
- 4, 5, 6 – drugi rząd,
- 7, 8, 9 – trzeci rząd,
- 1, 4, 7 – pierwsza kolumna,
- 2, 5, 8 – druga kolumna,
- 3, 6, 9 – trzecia kolumna,
- 1, 5, 9 – lewa przekątna,
- 3, 5, 7 – prawa przekątna.

Jeżeli znaki w którymkolwiek z tych zestawów pól będą takie same, to będzie to oznaczać, że wygrał jeden z graczy. Z racji tego, że będziemy sprawdzać te pola po każdym ruchu, to zawartość tych pól porównamy do znaku gracza, który właśnie wykonał ruch.

Do sprawdzenia tych pól moglibyśmy napisać dużą instrukcję `if`, która sprawdziłaby wszystkie pola:

```
return
  // pierwszy rząd
  (znaki[0] == znaki[1] && znaki[1] == znaki[2] && znaki[2] == znakGracza) ||
  // drugi rząd
  (znaki[3] == znaki[4] && znaki[4] == znaki[5] && znaki[5] == znakGracza) ||
  // trzeci rząd
  (znaki[6] == znaki[7] && znaki[7] == znaki[8] && znaki[8] == znakGracza) ||
  // pierwsza kolumna
  (znaki[0] == znaki[3] && znaki[3] == znaki[6] && znaki[6] == znakGracza) ||
  // druga kolumna
  (znaki[1] == znaki[4] && znaki[4] == znaki[7] && znaki[7] == znakGracza) ||
  // trzecia kolumna
  (znaki[2] == znaki[5] && znaki[5] == znaki[8] && znaki[8] == znakGracza) ||
  // lewa przekątna
  (znaki[0] == znaki[4] && znaki[4] == znaki[8] && znaki[8] == znakGracza) ||
  // prawa przekątna
  (znaki[2] == znaki[4] && znaki[4] == znaki[6] && znaki[6] == znakGracza);
```

Jednakże, ten kod jest mało czytelny z mojego punktu widzenia i zapisałem go w inny sposób – w metodzie `sprawdzCzyWygrana` zdefiniowałem dwuwymiarową tablicę, w której każda pod-tablica zawiera numery „wygrywających" pól, wylistowanych wcześniej (są one pomniejszone o `1`, bo stanowią indeksy w tablicy). Ta tablica wygląda następująco:

```
int[][] polaWygrywajace = {
    { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 }, // rzędy
    { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 }, // kolumny
    { 0, 4, 8 }, { 2, 4, 6 } // lewa i prawa przekątna
};
```

Teraz wystarczy, że w pętli przejdziemy przez tę tablicę i sprawdzimy, czy wszystkie pola wskazywane przez elementy danej pod-tablicy są takie same, jak znak aktualnego gracza:

```
for (int[] zestawPol : polaWygrywajace) {
  if (znaki[zestawPol[0]] == znaki[zestawPol[1]] &&
      znaki[zestawPol[1]] == znaki[zestawPol[2]] &&
      znaki[zestawPol[2]] == znakGracza) {
    return true;
  }
}
```

Jeżeli którykolwiek ze „zwycięskich” zestawów pól wskaże na trzy pola, która są wypełnione tym samym znakiem, to zwrócimy wartość `true`, która będzie świadczyć o wygranej gracza, który właśnie wykonał ruch. Jeżeli nie, to w linii `(15)` zwrócimy `false`, a gra będzie się toczyć dalej.

Pytania_i_zadania_rozwiazania/Rozdzial_10/KolkoIKrzyzyk.java

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

public class KolkoIKrzyzyk {
  public static void main(String[] args) {
    Character[] znaki = new Character[9];
    Arrays.fill(znaki, ' '); // 1
    int liczbaMozliwychRuchow = znaki.length; // 2
    char znakAktualnegoGracza = 'x'; // 3
    String wynik = null; // 4

    while (wynik == null) { // 5
      wyswietlPlansze(znaki); // 6
      int ruchGracza =
          pobierzRuchGracza(znakAktualnegoGracza);

      if (czyRuchPoprawny(ruchGracza, znaki)) { // 7
        znaki[ruchGracza] = znakAktualnegoGracza; // 8
        liczbaMozliwychRuchow--; // 9

        // 10
        if (sprawdzCzyWygrana(znaki, znakAktualnegoGracza)) {
          wynik = "Wygrał gracz " + znakAktualnegoGracza;
        } else if (liczbaMozliwychRuchow == 0) { // 11
          wynik = "Remis!";
        } else {
          znakAktualnegoGracza =
              znakAktualnegoGracza == 'x' ? 'o' : 'x'; // 12
        }
      }
    }

    // musimy jeszcze raz wyświetlić planszę,
    // aby pokazać ostatni ruch
    wyswietlPlansze(znaki); // 13
    System.out.println(wynik); // 14
  }
```

```
public static void wyswietlPlansze(Character[] znaki) {
  System.out.printf("""
      \n %s | %s | %s
      -----------
       %s | %s | %s
      -----------
       %s | %s | %s
      \n""", znaki);
}

public static int pobierzRuchGracza(char znakGracza) {
  System.out.println("Ruch gracza '" + znakGracza + "'");
  System.out.print("\nGdzie chcesz umieścić znak? [1-9] ");
  return pobierzLiczbe() - 1;
}

public static boolean czyRuchPoprawny(
    int ruchGracza, Character[] znaki
) {
  if (ruchGracza < 0 || ruchGracza >= znaki.length) {
    System.out.println("\n----- Nieprawidłowy ruch.\n");
    return false;
  } else if (znaki[ruchGracza] != ' ') {
    System.out.println("\n----- To pole jest już zajęte.\n");
    return false;
  }
  return true;
}

public static boolean sprawdzCzyWygrana(
    Character[] znaki, char znakGracza
) {
  int[][] polaWygrywajace = {
      { 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 }, // rzędy
      { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 }, // kolumny
      { 0, 4, 8 }, { 2, 4, 6 } // lewa i prawa przekątna
  };
  for (int[] zestawPol : polaWygrywajace) {
    if (znaki[zestawPol[0]] == znaki[zestawPol[1]] &&
        znaki[zestawPol[1]] == znaki[zestawPol[2]] &&
        znaki[zestawPol[2]] == znakGracza) {
      return true;
    }
  }
  return false; // 15
}
```

```
  public static int pobierzLiczbe() {
    return new Scanner(System.in).nextInt();
  }
}
```

Przykładowe uruchomienie tego programu:

```
> java KolkoIKrzyzyk

   |   |
-----------
   |   |
-----------
   |   |

Ruch gracza 'x'

Gdzie chcesz umieścić znak? [1-9] 1

 x |   |
-----------
   |   |
-----------
   |   |

Ruch gracza 'o'

Gdzie chcesz umieścić znak? [1-9] 2

 x | o |
-----------
   |   |
-----------
   |   |

Ruch gracza 'x'

Gdzie chcesz umieścić znak? [1-9] 3

 x | o | x
-----------
   |   |
-----------
   |   |

Ruch gracza 'o'
```

```
Gdzie chcesz umieścić znak? [1-9] 5

 x | o | x
-----------
   | o |
-----------
   |   |

Ruch gracza 'x'

Gdzie chcesz umieścić znak? [1-9] 8

 x | o | x
-----------
   | o |
-----------
   | x |

Ruch gracza 'o'

Gdzie chcesz umieścić znak? [1-9] 4

 x | o | x
-----------
 o | o |
-----------
   | x |

Ruch gracza 'x'

Gdzie chcesz umieścić znak? [1-9] 6

 x | o | x
-----------
 o | o | x
-----------
   | x |

Ruch gracza 'o'

Gdzie chcesz umieścić znak? [1-9] 9

 x | o | x
-----------
 o | o | x
-----------
   | x | o
```

```
Ruch gracza 'x'

Gdzie chcesz umieścić znak? [1-9] 7

 x | o | x
-----------
 o | o | x
-----------
 x | x | o

Remis!
```