Zbiór zadań - Java

Piotr Jastrzębski

2025-10-07

Spis treści

Zbiór zadań - Java

Tu będzie zbiór zadań z programowania w języku Java. Inspiracją było zebranie zadań powstałych w trakcie prowadzenia zajęć dydaktycznych realizowanych na Wydziale Matematyki i Informatyki Uniwersytetu Warmińsko-Mazurskiego w Olsztynie.

Część I

Wprowadzenie do języka Java

1 Operacje wejścia/wyjścia

1. Napisz prostą aplikację kalkulatora tekstowego, która przyjmuje dwa liczby od użytkownika jako wejście i wykonuje podstawowe operacje matematyczne (dodawanie, odejmowanie, mnożenie, dzielenie). Wyświetl wyniki na ekranie.
2. Napisz program, który wczytuje ze standardowego wejścia dwa łańcuchy znaków, a następnie wypisuje je w kolejnych wierszach na standardowym wyjściu.
3. Napisz program, który wczytuje ze standardowego wejścia cztery liczby wymierne, a następnie wypisuje ich sumę na standardowym wyjściu.
4. Stwórz program do obliczenia pola kwadratu. Dane pobierz od użytkownika, wynik wyświetl na standardowym wyjściu.
5. Napisz program, w którym wartości zmiennych pobierasz od użytkownika i wykonasz i wyświetlisz wyniki operacji:

- $a + b - x^2$
- $\frac{a-b}{c-3}$
- $3(4 + 5a)(b - c^3)$

6. Napisz program, w którym zostaną wykonane poniższe operacje za pomocą tzw. złożonych operatorów przypisania (`+=` , `-=` i innych podobnych operatorów):

- $a = a + 4$
- $b = b - a$
- $c = c(2 - 4a)$
- $d = \frac{d}{4-a^2}$

2 Instrukcje warunkowe

1. Napisz program, który sprawdza, czy podana liczba całkowita jest parzysta. Jeżeli tak, program powinien wypisać "Liczba jest parzysta", w przeciwnym razie "Liczba jest nieparzysta".
2. Napisz program, który przyjmuje trzy liczby całkowite jako argumenty i zwraca największą z nich. Zastosuj instrukcje warunkowe do porównania liczb.
3. Napisz program, który na podstawie podanego jako argument numeru dnia tygodnia (od 1 do 7) wypisze nazwę tego dnia tygodnia. Dla przykładu, jeżeli użytkownik poda liczbę 1, program powinien wypisać "Poniedziałek". Jeżeli podana liczba nie jest z zakresu od 1 do 7, program powinien wyświetlić komunikat "Niepoprawny numer dnia tygodnia".
4. Napisz program, który rozwiązuje równanie kwadratowe o postaci $ax^2 + bx + c = 0$. Program powinien przyjmować jako argumenty wartości a, b, i c, obliczać delty ($b^2 - 4ac$), a następnie zwracać rozwiązania równania w zależności od wartości delty.
5. Napisz program, który przyjmuje wiek użytkownika jako argument. Jeżeli wiek jest mniejszy niż 18, program powinien wyświetlić "Jesteś niepełnoletni". Jeżeli wiek jest większy lub równy 18, ale mniejszy od 65, program powinien wyświetlić "Jesteś dorosły". Jeżeli wiek jest równy lub większy niż 65, program powinien wyświetlić "Jesteś emerytem".
6. Napisz program, który będzie sprawdzał, czy podany rok jest rokiem przestępnym. Rok jest przestępny, jeśli jest podzielny przez 4, ale nie jest podzielny przez 100, chyba że jest podzielny przez 400.
7. Napisz program, który przyjmuje trzy liczby całkowite jako argumenty i sortuje je w kolejności rosnącej, używając instrukcji warunkowych, a następnie wyświetla posortowane liczby.
8. Napisz program, który oblicza podatek dochodowy na podstawie podanych dochodów i zasad podatkowych. Załóżmy, że podatek wynosi 18% dla dochodu do 85,528 PLN, a dla dochodu powyżej tej kwoty podatek wynosi 14,839.02 PLN plus 32% nadwyżki ponad 85,528 PLN. Użytkownik powinien wprowadzić swoje dochody, a program powinien obliczyć i wyświetlić kwotę podatku.
9. Napisz program sprawdzający czy podane liczby z klawiatury mogą stanowić poprawną datę w kalendarzu.

Przykładowe wejście:

```
Podaj dzień: 29
Podaj miesiąc: 2
Podaj rok: 2017
```

Przykładowe wyjście:

```
Błędna data
```

10. Napisz program sprawdzający czy podane liczby z klawiatury mogą stanowić poprawna godzinę w formacie 24-godzinnym.

Przykładowe wejście:

```
Podaj godzinę: 22
Podaj minuty: 12
Podaj sekundy: 33
```

Przykładowe wyjście:

```
Poprawna godzina!
```

11. Napisz program, w którym użytkownik ma wprowadzić trzycyfrową liczbę całkowitą. Następnie należy sprawdzić czy liczba jest palindromem. Stosowny komunikat wyświetl na konsoli.
12. Napisz program, który pobiera trzy liczby całkowite (teoretycznie mogą być różnych znaków) i sprawdza, czy można z nich zbudować trójkąt prostokątny - ostatecznie wypisuje „TAK” lub „NIE”.

3 Pętle

1. Napisz program, który wykorzystując pętlę `for` wyświetli liczby od 1 do 100.
2. Napisz program, który przy użyciu pętli `while` obliczy sumę liczb od 1 do 50.
3. Napisz program w Javie, który za pomocą pętli `for` generuje pierwsze 10 liczb ciągu Fibonacciego.
4. Stwórz program, który używając zagnieżdżonych pętli `for`, wyświetli tabliczkę mnożenia dla liczb od 1 do 10.
5. Napisz program, który używając pętli `do-while`, wyświetli pierwsze 20 liczb parzystych i nieparzystych.
6. Napisz program, który sprawdzi, czy podana liczba jest liczbą pierwszą. Liczba powinna być wprowadzona przez użytkownika.
7. Napisz program, który wczyta od użytkownika ciąg 10 liczb całkowitych, a następnie wyświetli największą i najmniejszą z nich.
8. Napisz program, który oblicza sumę cyfr dowolnej wprowadzonej liczby. Program powinien akceptować liczbę jako input od użytkownika.
9. Napisz program, który generuje i wyświetla pierwsze 10 elementów szeregu geometrycznego o zadanym pierwszym elemencie i ilorazie. Parametry szeregu powinny być wprowadzane przez użytkownika.
10. Stwórz program, który przyjmie od użytkownika liczbę całkowitą i zwróci tę liczbę w odwrotnej kolejności. Na przykład, dla liczby 12345, wynik powinien wynosić 54321. Możesz ograniczyć program tylko do liczb dodatnich.
11. Napisz program, który obliczy sumę kwadratów liczb od 1 do n, gdzie n jest liczbą wprowadzoną przez użytkownika.
12. Napisz program, który znajdzie i wyświetli wszystkie liczby doskonałe mniejsze od 10 000. Liczba doskonała to taka, której suma dzielników (bez niej samej) jest równa jej wartości. Na przykład, 6 jest liczbą doskonałą, ponieważ $1 + 2 + 3 = 6$.
13. Napisz program, który znajdzie wszystkie liczby Amstronga mniejsze od 10 000. Liczba Amstronga to taka, której suma jej cyfr podniesionych do potęgi równiej liczbie cyfr w tej liczbie, jest równa samej liczbie. Na przykład 153 jest liczbą Amstronga, ponieważ $1^3 + 5^3 + 3^3 = 153$.

14. Napisz program, który dla dwóch podanych liczb obliczy ich najmniejszą wspólną wielokrotność (NWW). Użytkownik powinien podać dwie liczby jako dane wejściowe.

15. Napisz program pobierający z klawiatury liczbę całkowitą dodatnią. Następnie narysuj odpowiedni trójkąt np. dla 5:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

15. Napisz program, który pobiera od użytkownika liczbę naturalną n ($n \geqslant 0$). Następnie wyznacz i wyświetl n-ty element ciągu:

a) $(4, -8, 16, -32, 64, ...)$
b) $(2, 6, 18, 54, 162, ...)$
c) $(8, 3, -2, -7, -12, ...)$

16. Napisz program wczytujący kolejne liczby całkowite (różnych znaków) z klawiatury i kończący się gdy ich suma przekroczy 100.

17. Napisz program, który pobiera od użytkownika dodatnią liczbę całkowitą n, następnie n liczb całkowitych $a_1, ..., a_n$. Program ma wyświetlić ile spośród tych liczb spełnia warunek $3^k < a_k < k!$ dla $1 \leq k \leq n$.

18. Napisać program wyświetlający na ekranie pierwsze szesnaście potęg dwójki. Wykorzystaj w tym celu pętle.

19. Napisz program, który pobiera od użytkownika 5 liczb całkowitych. Pobieranie ma zostać przerwane, gdy użytkownik wprowadzić liczbę ujemną. Jeśli pobieranie nie zostanie przerwane, wyświetl sumę wprowadzonych liczb.

20. Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite n i m (zakładamy, że $n < m$) i wypisuje na standardowym wyjściu wartość liczby $n \cdot ... \cdot m$.

21. Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite n i m (zakładamy, że $n < m$) i wypisuje na standardowym wyjściu wartość liczby $n + ... + m$.

22. Napisz program, który pobiera od użytkownika trzy dodatnie liczby całkowite a, b, c. Na standardowym wyjściu wyświetl dodatnie liczby całkowite większe od b, mniejsze lub równe od a i podzielne przez c.

23. Napisz program, który ze standardowego wejścia pobiera liczbę naturalną a a następnie wypisuje na standardowym wyjściu ile z cyfr liczby a jest równe 7.

24. Napisać program, który wczyta z wejścia liczby całkowite aż do napotkania liczby ujemnej, a następnie wyświetla największy oraz najmniejszy element z wczytanych liczb (z pominięciem ostatniej, ujemnej liczby).

25. Napisz program, który prosi użytkownika o wprowadzenie dodatniej liczby całkowitej `n`, a następnie oblicza i wyświetla największą liczbę naturalną `k`, taką że $k!$ (silnia) nie przekracza `n`. Zadanie należy rozwiązać bez tablic, napisów, wbudowanych funkcji matematycznych. Nie twórz samodzielnie też własnych funkcji.

Przykład: dla `n=25` powinno być wyświetlone 4 (bo $4! = 24 \leqslant 25$, ale $5! = 120 > 25$), zaś dla `n=120` powinno być wyświetlone 5.

4 Funkcje/metody

1. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n. Metoda ma zwrócić jako liczbę całkowitą sumę szeregu

$$1 - 2 + 3 - 4 + ... \pm n.$$

Stwórz przypadek testowy dla tej metody.

2. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n ($n > 2$). Metoda ma zwrócić największą liczbę pierwszą mniejszą niż n. Stwórz przypadek testowy dla tej metody.

3. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n. Metoda zwraca odpowiednią wartość logiczną sprawdzającą czy n jest liczbą doskonałą. Liczba doskonała to taka, której suma dzielników jest równa tej liczbie (liczbami doskonałymi są np. $1 = 1$, $6 = 1 + 2 + 3$). Stwórz przypadek testowy dla metody.

4. Napisz statyczną metodę, której parametrami są dwie dodatnie liczby całkowite a i b. Metoda ma zwrócić najmniejszą wspólną wielokrotność (NWW) liczb a i b. Stwórz przypadek testowy dla metody.

Przykład: $NWW(5, 10) = 10$, $NWW(4, 5) = 20$.

5. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n. Metoda zwraca `true` jeśli zadana liczba n jest nieparzysta, dodatnia, składa się z 4 cyfr i podzielna przez 5, oraz zwraca `false` w pozostałych przypadkach. Stwórz przypadek testowy dla metody.

6. Napisz statyczną metodę, która jako argument otrzymuje dodatnią liczbę całkowitą n i zwraca liczbę 7^{-n}. Nie korzystaj z żadnych gotowych funkcji bibliotecznych ani wbudowanych wewnątrz tej funkcji poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

Podpowiedź: $7^{-n} = \frac{1}{7^n}$.

7. Napisz statyczną metodę, której argumentem są cztery dodatnie liczby całkowite a, b, c, d. Metoda zwraca ile liczb całkowitych z przedziału (a, b) jest podzielnych przez c i nie jest podzielnych przez d. W przypadku braku takich liczb, zwróć zero. Stwórz przypadek testowy dla tej metody.

8. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n. Metoda zwraca sumę liczb całkowitych od n do $2n$ (włącznie). Stwórz przypadek testowy dla metody.

9. Napisz statyczną metodę, której argumentem są nieujemne liczby całkowite n i k. Metoda zwraca wartość wyrażenia:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Stwórz przypadek testowy dla metody.

10. Napisz statyczną metodę, której argumentem jest dodatnia liczba całkowita n. Metoda zwraca 1 jeśli n jest liczbą składającą się z samych jedynek w zapisie dziesiętnym oraz zwraca 0 w przeciwnym wypadku. Stwórz przypadek testowy dla metody.

11. Napisz statyczną rekurencyjną metodę `quadSeq`, która dla otrzymanej w argumencie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób:

$$v_0 = 1$$

$$v_n = \begin{cases} v_{n-1} + n, & \text{jeśli } n \text{ jest podzielne przez 4} \\ v_{n-1} \cdot 2, & \text{jeśli } n \text{ daje resztę 1 z dzielenia przez 4} \\ v_{n-1}^2 - 1, & \text{jeśli } n \text{ daje resztę 2 z dzielenia przez 4} \\ \lceil \frac{v_{n-1}}{3} \rceil, & \text{jeśli } n \text{ daje resztę 3 z dzielenia przez 4} \end{cases}$$

gdzie $\lceil x \rceil$ oznacza funkcję sufit (najmniejsza liczba całkowita nie mniejsza od x). W rozwiązaniu nie korzystać ze wbudowanych funkcji matematycznych. Stwórz przypadek testowy dla funkcji.

n	0	1	2	3	4	5	6
v_n	1	2	3	1	5	10	99

5 Generowanie liczb pseudolosowych

1. Napisz metodę `generateRandomInt`, która generuje i zwraca losową liczbę całkowitą. Stwórz przypadek testowy.
2. Napisz metodę `generateRandomDouble`, która generuje i zwraca losową liczbę zmiennoprzecinkową z zakresu od 0.0 do 1.0. Stwórz przypadek testowy.
3. Napisz metodę `generateRandomIntInRange`, która przyjmuje dwie liczby całkowite jako argumenty i zwraca losową liczbę całkowitą z tego zakresu (włącznie z granicami). Na przykład, dla argumentów 5 i 10, metoda powinna zwracać liczbę z zakresu od 5 do 10. Stwórz przypadek testowy.
4. Napisz metodę `generateRandomGaussian`, która generuje i zwraca losową liczbę zmiennoprzecinkową zgodnie z rozkładem normalnym. Stwórz przypadek testowy.
5. Napisz metodę `generateRandomBoolean`, która generuje i zwraca losową wartość logiczną (true lub false). Stwórz przypadek testowy.

6 Tablice

1. Napisz program, który tworzy tablicę jednowymiarową 10 liczb całkowitych, a następnie wyświetla je w konsoli w porządku odwrotnym do wprowadzenia.
2. Utwórz program, który tworzy jednowymiarową tablicę 20 liczb losowych z przedziału od 1 do 100, a następnie oblicza i wyświetla ich średnią wartość.
3. Napisz program, który tworzy tablicę jednowymiarową 15 liczb całkowitych, a następnie oblicza i wyświetla największą i najmniejszą wartość w tablicy.
4. Utwórz program, który tworzy jednowymiarową tablicę 30 liczb całkowitych. Następnie poproś użytkownika, aby podał dowolną liczbę. Program powinien wyświetlić informację, czy podana liczba znajduje się w tablicy, a także ile razy się w niej pojawia.
5. Napisz program, który tworzy jednowymiarową tablicę 10 liczb całkowitych. Program powinien obliczać i wyświetlać sumę tych liczb, które są parzyste.
6. Napisz program, który tworzy jednowymiarową tablicę 10 liczb zmiennoprzecinkowych, a następnie oblicza i wyświetla ich sumę.
7. Utwórz program, który tworzy tablicę jednowymiarową 20 liczb zmiennoprzecinkowych, a następnie znajduje i wyświetla wartość średnią oraz medianę tych liczb.
8. Napisz program, który tworzy jednowymiarową tablicę 10 liczb zmiennoprzecinkowych. Program powinien obliczać i wyświetlać sumę tych liczb, które są większe niż 0.5.
9. Napisz program, który tworzy jednowymiarową tablicę 20 liczb całkowitych, a następnie oblicza i wyświetla ilość liczb parzystych i nieparzystych w tablicy.
10. Utwórz program, który tworzy jednowymiarową tablicę 30 liczb całkowitych. Następnie program powinien obliczyć i wyświetlić ilość liczb, które są kwadratami innej liczby całkowitej.
11. Napisz program, który tworzy jednowymiarową tablicę 50 liczb całkowitych. Program powinien obliczać i wyświetlać ilość liczb, które są liczbami pierwszymi.
12. Napisz program, który tworzy jednowymiarową tablicę 15 liczb całkowitych, a następnie oblicza i wyświetla sumę liczb, które są podzielne przez 3.

13. Napisz statyczną metodę `minimumValue`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca najmniejszą liczbę w tablicy. Przyjmij, że tablica zawsze będzie miała co najmniej jeden element. Jeżeli tablica zawiera tylko jeden element, zwróć ten element. Stwórz przypadek testowy.

14. Napisz statyczną metodę `average`, która przyjmuje tablicę liczb zmiennoprzecinkowych jako argument i zwraca średnią arytmetyczną wszystkich liczb w tablicy. Jeżeli tablica jest pusta, zwróć 0. Stwórz przypadek testowy.

15. Napisz statyczną metodę `reverseArray`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca nową tablicę, ale z odwróconym porządkiem elementów. Na przykład, dla tablicy [1, 2, 3, 4, 5], funkcja powinna zwrócić [5, 4, 3, 2, 1]. Stwórz przypadek testowy.

16. Napisz statyczną metodę `reverseArray`, która przyjmuje tablicę liczb całkowitych jako argument. Metoda odwraca porządek elementów w tablicy i powinna być procedurą. Na przykład, dla tablicy [1, 2, 3, 4, 5], funkcja powinna zmienić tablicę na [5, 4, 3, 2, 1]. Stwórz przypadek testowy.

17. Napisz statyczną metodę `countZeros`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca liczbę wystąpień zera w tablicy. Na przykład, dla tablicy [0, 1, 2, 0, 3, 0, 4], funkcja powinna zwrócić 3. Stwórz przypadek testowy.

18. Napisz statyczną metodę `oddElementsSum`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca sumę wszystkich nieparzystych liczb w tablicy. Jeżeli w tablicy nie ma żadnych nieparzystych liczb, funkcja powinna zwrócić 0. Stwórz przypadek testowy.

19. Napisz statyczną metodę `copyArray`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca nową tablicę, która jest kopią pierwotnej. Funkcja powinna skopiować tablicę ręcznie, element po elemencie, bez korzystania z systemowych metod kopiowania. Stwórz przypadek testowy.

20. Napisz statyczną metodę `copyArrayReverse`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca nową tablicę, która jest odwróconą kopią pierwotnej. Na przykład, dla tablicy [1, 2, 3, 4, 5], funkcja powinna zwrócić [5, 4, 3, 2, 1]. Stwórz przypadek testowy.

21. Napisz statyczną metodę `copyArrayEven`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca nową tablicę zawierającą tylko parzyste liczby z pierwotnej tablicy. Na przykład, dla tablicy [1, 2, 3, 4, 5], funkcja powinna zwrócić [2, 4]. Jeżeli w tablicy nie ma żadnych parzystych liczb, funkcja powinna zwrócić pustą tablicę. Stwórz przypadek testowy.

22. Napisz statyczną metodę `copyArrayWithIndex`, która przyjmuje tablicę liczb całkowitych oraz indeks początkowy i końcowy jako argumenty. Funkcja powinna zwrócić nową tablicę zawierającą elementy pierwotnej tablicy od indeksu początkowego do końcowego (włącznie). Na przykład, dla tablicy [1, 2, 3, 4, 5] i indeksów 1 i 3, funkcja powinna zwrócić [2, 3, 4]. Stwórz przypadek testowy.

23. Napisz statyczną metodę `mergeArrays`, która przyjmuje dwie tablice liczb całkowitych jako argumenty. Funkcja powinna zwrócić nową tablicę, która jest połączeniem obu pierwotnych tablic. Na przykład, dla tablic [1, 2, 3] i [4, 5, 6], funkcja powinna zwrócić [1, 2, 3, 4, 5, 6]. Stwórz przypadek testowy.

24. Napisz metodę `sortArray`, która przyjmuje tablicę liczb całkowitych jako argument i zwraca nową tablicę, która jest posortowaną wersją pierwotnej tablicy. Wykorzystaj metodę `Arrays.sort()` z biblioteki `java.util.Arrays` do posortowania tablicy. Stwórz przypadek testowy.

25. Napisz metodę `checkEquality`, która przyjmuje dwie tablice liczb całkowitych jako argumenty i zwraca wartość `true`, jeśli tablice są równe, a `false` w przeciwnym razie. Wykorzystaj metodę `Arrays.equals()` z biblioteki `java.util.Arrays` do porównania tablic. Stwórz przypadek testowy.

26. Napisz metodę `fillArray`, która przyjmuje tablicę liczb całkowitych i liczbę całkowitą jako argumenty. Metoda powinna wypełnić tablicę podaną liczbą, wykorzystując do tego metodę `Arrays.fill()` z biblioteki `java.util.Arrays`. Stwórz przypadek testowy.

27. Napisz metodę `printArray`, która przyjmuje tablicę liczb całkowitych jako argument i drukuje jej zawartość na konsoli. Wykorzystaj do tego metodę `Arrays.toString()` z biblioteki `java.util.Arrays`, która zwraca tekstową reprezentację tablicy. Stwórz przypadek testowy.

28. Napisz metodę `copyArray`, która przyjmuje tablicę liczb wymiernych jako argument. Metoda powinna zwracać nową tablicę, będącą kopią przekazanej tablicy. Do skopiowania tablicy wykorzystaj metodę `Arrays.copyOf()` z biblioteki `java.util.Arrays`. Stwórz przypadek testowy.

29. Napisz metodę `sortArray`, która przyjmuje tablicę liczb wymiernych jako argument. Metoda powinna sortować tablicę w porządku rosnącym, wykorzystując do tego metodę `Arrays.sort()` z biblioteki `java.util.Arrays`. Stwórz przypadek testowy.

7 ArrayList (listy tablicowe)

1. Napisz program, który tworzy listę tablicową 10 liczb całkowitych, a następnie wyświetla je w konsoli w porządku odwrotnym do wprowadzenia.
2. Napisz program, który tworzy listę tablicową 10 liczb zmiennoprzecinkowych, a następnie oblicza i wyświetla ich sumę.
3. Napisz statyczną metodę `minimumValue`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca najmniejszą liczbę w liście tablicowej. Przyjmij, że lista tablicowa zawsze będzie miała co najmniej jeden element. Stwórz przypadek testowy.
4. Napisz statyczną metodę `minimumValue`, która przyjmuje listę tablicową liczb wymiernych jako argument i zwraca najmniejszą liczbę w liście tablicowej. Przyjmij, że lista tablicowa zawsze będzie miała co najmniej jeden element. Stwórz przypadek testowy.
5. Napisz statyczną metodę `average`, która przyjmuje listę tablicową liczb zmiennoprzecinkowych jako argument i zwraca średnią arytmetyczną wszystkich liczb. Jeżeli lista tablicowa jest pusta, zwróć 0. Stwórz przypadek testowy.
6. Napisz statyczną metodę `reverseArray`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca nową listę tablicową, ale z odwróconym porządkiem elementów. Na przykład, dla [1, 2, 3, 4, 5], funkcja powinna zwrócić [5, 4, 3, 2, 1]. Stwórz przypadek testowy.
7. Napisz statyczną metodę `reverseArray`, która przyjmuje listę tablicową liczb całkowitych jako argument. Metoda odwraca porządek elementów w liście tablicowej i powinna być procedurą. Na przykład, dla [1, 2, 3, 4, 5], funkcja powinna zmienić listę tablicową na [5, 4, 3, 2, 1]. Stwórz przypadek testowy.
8. Napisz statyczną metodę `countZeros`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca liczbę wystąpień zera w liście tablicowej. Na przykład, dla [0, 1, 2, 0, 3, 0, 4], funkcja powinna zwrócić 3. Stwórz przypadek testowy.
9. Napisz statyczną metodę `oddElementsSum`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca sumę wszystkich nieparzystych liczb w liście tablicowej. Jeżeli w liście tablicowej nie ma żadnych nieparzystych liczb, funkcja powinna zwrócić 0. Stwórz przypadek testowy.

10. Napisz statyczną metodę `copyArray`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca nową listę tablicową, która jest kopią pierwotnej. Funkcja powinna skopiować listę tablicową ręcznie, element po elemencie, bez korzystania z systemowych metod kopiowania. Stwórz przypadek testowy.

11. Napisz statyczną metodę `copyArrayReverse`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca nową listę tablicową, która jest odwróconą kopią pierwotnej. Na przykład, dla liście tablicowej [1, 2, 3, 4, 5], funkcja powinna zwrócić [5, 4, 3, 2, 1]. Stwórz przypadek testowy.

12. Napisz statyczną metodę `copyArrayEven`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca nową listę tablicową zawierającą tylko parzyste liczby z pierwotnej liście tablicowej. Na przykład, dla liście tablicowej [1, 2, 3, 4, 5], funkcja powinna zwrócić [2, 4]. Jeżeli w liście tablicowej nie ma żadnych parzystych liczb, funkcja powinna zwrócić pustą listę tablicową. Stwórz przypadek testowy.

13. Napisz statyczną metodę `copyArrayWithIndex`, która przyjmuje listę tablicową liczb całkowitych oraz indeks początkowy i końcowy jako argumenty. Funkcja powinna zwrócić nową listę tablicową zawierającą elementy pierwotnej liście tablicowej od indeksu początkowego do końcowego (włącznie). Na przykład, dla liście tablicowej [1, 2, 3, 4, 5] i indeksów 1 i 3, funkcja powinna zwrócić [2, 3, 4]. Stwórz przypadek testowy.

14. Napisz statyczną metodę `mergeArrays`, która przyjmuje dwie tablice liczb całkowitych jako argumenty. Funkcja powinna zwrócić nową listę tablicową, która jest połączeniem obu pierwotnych tablic. Na przykład, dla list tablicowych [1, 2, 3] i [4, 5, 6], funkcja powinna zwrócić [1, 2, 3, 4, 5, 6]. Stwórz przypadek testowy.

15. Napisz metodę `sortArray`, która przyjmuje listę tablicową liczb całkowitych jako argument i zwraca nową listę tablicową, która jest posortowaną wersją pierwotnej liście tablicowej. Stwórz przypadek testowy.

8 Napisy

1. Napisz statyczną metodę, która przyjmuje napis jako argument i zwraca ten napis w odwrotnej kolejności. Stwórz przypadek testowy.
2. Napisz statyczną metodę, która sprawdza, czy dany napis jest palindromem. Palindrom to słowo, fraza, liczba lub inny ciąg znaków, który czyta się tak samo od przodu, jak i od tyłu. Stwórz przypadek testowy.
3. Napisz statyczną metodę, która usuwa wszystkie spacje z danego napisu. Stwórz przypadek testowy.
4. Napisz statyczną metodę, która przyjmuje napis jako argument i zwraca ten napis z zamienioną pierwszą i ostatnią literą. Stwórz przypadek testowy.
5. Napisz program, który analizuje dany napis pod kątem częstotliwości występowania każdego ze znaków. Program powinien wyświetlić znak i liczbę jego wystąpień w danym napisie. Dane pobierz ze standardowego wejścia.
6. Napisz program, który przyjmuje napis jako wejście i wypisuje wszystkie znaki znajdujące się na parzystych indeksach napisu, używając metody `charAt`.
7. Używając metody `charAt`, napisz statyczną metodę, która sprawdza, czy dany napis zaczyna się i kończy tym samym znakiem. Stwórz przypadek testowy.
8. Napisz program, który przyjmuje trzy napisy: główny napis, prefiks i sufiks. Używając metod `startsWith` oraz `endsWith`, sprawdź czy główny napis zaczyna się od podanego prefiksu i kończy podanym sufiksem. Wypisz odpowiedni komunikat dla każdego z tych przypadków.
9. Napisz program, który przyjmuje jako wejście pojedynczy znak oraz liczbę całkowitą `n`. Używając klasy `StringBuilder`, zbuduj i wypisz piramidę o wysokości `n`, gdzie każdy poziom piramidy składa się z podanego znaku. Na przykład dla znaku `*` i `n=3`, oczekiwany wynik to:

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

10. Dany jest napis. Używając `StringBuilder`, napisz program, który usuwa wszystkie powtarzające się znaki, pozostawiając tylko pierwsze wystąpienie każdego znaku. Na przykład dla napisu "bananowy", oczekiwany wynik to "banowy".

11. Napisz program, który przyjmuje zdanie jako wejście. Używając `StringBuilder`, odwróć każde słowo w zdaniu, ale zachowaj kolejność słów. Na przykład dla zdania "Java jest fajna", oczekiwany wynik to "avaJ tsej anjaf".

12. Napisz metodę statyczną `reverseString`, która przyjmuje jako argument obiekt typu `StringBuilder` i zwraca nowy `StringBuilder`, będący odwróconym napisem pierwotnym. Stwórz przypadek testowy.

13. Napisz metodę statyczną `removeDuplicates`, która przyjmuje `StringBuilder` i usuwa wszystkie powtarzające się znaki w napisie, pozostawiając tylko pierwsze wystąpienie danego znaku. Stwórz przypadek testowy.

14. Napisz metodę statyczną `mostFrequentChar`, która przyjmuje `StringBuilder` jako argument i zwraca znak, który występuje najczęściej w napisie. W przypadku remisów, zwróć pierwszy znak z remisowych. Stwórz przypadek testowy.

15. Napisz metodę statyczną `insertSubstring`, która przyjmuje dwa argumenty: `StringBuilder sb` oraz `String toInsert`. Metoda ma wstawiać `toInsert` w środek pierwotnego `StringBuilder`. Stwórz przypadek testowy.

16. Napisz metodę statyczną `splitByEvenOdd`, która przyjmuje `StringBuilder` jako argument. Metoda powinna zwracać tablicę dwóch elementów typu `StringBuilder`. Pierwszy element tablicy ma zawierać znaki z nieparzystych indeksów pierwotnego napisu, a drugi z parzystych. Stwórz przypadek testowy.

17. Napisz metodę statyczną `capitalizeEverySecond`, która przyjmuje jako argument obiekt typu `StringBuffer`. Metoda ma zmienić każdą drugą literę na wielką. Stwórz przypadek testowy.

18. Napisz metodę statyczną `replaceSubstring`, która przyjmuje dwa argumenty: `StringBuffer sb` i `String oldSub`, oraz `String newSub`. Metoda ma zamienić wszystkie wystąpienia podciągu `oldSub` na `newSub`. Stwórz przypadek testowy.

19. Napisz metodę statyczną `countOccurrences`, która przyjmuje dwa argumenty: `StringBuffer sb` oraz `char c`. Metoda powinna zwracać liczbę wystąpień znaku `c` w napisie. Stwórz przypadek testowy.

20. Napisz metodę statyczną `trimToSize`, która przyjmuje `StringBuffer sb` i liczbę całkowitą `n`. Metoda ma zmniejszyć długość napisu do `n` znaków (jeśli pierwotny napis jest dłuższy). Jeśli napis jest krótszy lub równy `n`, nie powinien ulec zmianie. Stwórz przypadek testowy.

21. Napisz metodę statyczną `isPalindrome`, która przyjmuje `StringBuffer` jako argument. Metoda powinna sprawdzić, czy napis jest palindromem (odczytywany tak samo od przodu jak i od tyłu) i zwrócić odpowiednią wartość logiczną. Stwórz przypadek testowy.

Część II

Programowanie obiektowe

9 Pojęcie klasy/obiektu

1. Utwórz klasę `Dog` z polami: `name`, `breed` i `age`. Napisz metodę `bark()`, która wydrukuje na konsoli `"Wow Wow"`. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.

2. Stwórz klasę `Car` z polami: `brand`, `model` i `speed`. Napisz metody `accelerate(int value)` i `decelerate(int value)`, które odpowiednio zwiększają i zmniejszają prędkość o podaną wartość. Stwórz przypadek testowy, aby wywołać każdą metodę co najmniej jeden raz.

3. Stwórz klasę `BankAccount` z polem `balance`. Napisz metody `deposit(double amount)` i `withdraw(double amount)`, które odpowiednio zwiększają i zmniejszają saldo o daną kwotę. Stwórz przypadek testowy, aby wywołać każdą metodę co najmniej jeden raz.

4. Utwórz klasę `Point` z dwoma polami: `x` i `y` reprezentującymi współrzędne na płaszczyźnie. Napisz metodę `distance(Point otherPoint)`, która oblicza odległość między bieżącym punktem a innym punktem. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.

5. Stwórz klasę `Time` z polami: `hours` i `minutes`. Napisz metodę `addTime(Time otherTime)`, która dodaje do bieżącego czasu czas podany jako argument i zwraca nowy obiekt klasy Time. Zadbaj o to, aby minuty i godziny nie przekraczały odpowiednio 59 i 23. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.

6. Wykonaj po kolei dwa podpunkty:

   A. Stwórz klasę `Car` zawierającą publiczne pole `brand`. W klasie `TestCar`, utwórz obiekt klasy `Car`, przypisz mu wartość `null` i spróbuj odwołać się do pola `brand`. Jaki jest wynik?

   B. Zmodyfikuj kod z podpunktu A tak, aby sprawdzić, czy obiekt `Car` jest `null` przed odwołaniem się do pola `brand`. Jaki jest wynik?

7. Utwórz klasę `Person` zawierającą pole `name`. W klasie `TestPerson`, utwórz dwa obiekty klasy `Person` - `person1` i `person2` - oba odnoszące się do tego samego obiektu. Zmień wartość pola `name` przez `person1` i wydrukuj wartość pola `name` przez `person2`. Przeanalizuj wynik.

8. Stwórz tablicę obiektów klasy `Car` (klasa zawiera publiczne pole `brand`). Następnie spróbuj odwołać się do pola `brand` jednego z obiektów w tablicy, nie inicjalizując wcześniej tablicy obiektami `Car`. Przeanalizuj wynik.

9. Stwórz listę tablicową (ArrayList) obiektów klasy `Person` (klasa zawiera publiczne pole `name`). Następnie spróbuj odwołać się do pola `name` jednego z obiektów na liście, nie dodając wcześniej do listy żadnych obiektów `Person`. Przeanalizuj wynik.

10. Utwórz klasę `Dog` z metodą `bark`, która wydrukuje wiadomość "Woof! Woof!". W klasie `TestDog` utwórz obiekt `Dog`, przypisz mu wartość `null` i spróbuj wywołać metodę `bark`. Przeanalizuj wynik.

11. Stwórz klasę `Cat` z polem `name`. Dodaj do klasy `Cat` metodę `createCat`, która zwraca nowy obiekt klasy `Cat`. Metoda powinna ustawiać pole `name` na podaną wartość, ale tylko jeśli wartość nie jest `null`. W przeciwnym razie powinna zwracać `null`. W klasie `TestCat` użyj metody `createCat` do stworzenia obiektu `cat`, a następnie spróbuj wydrukować wartość pola `name`. Co się stanie, jeśli przekażesz `null` jako argument do metody `createCat`? Przeanalizuj wynik.

12. Wykonaj kolejno poniższe czynności:

    A. Stwórz klasę `Person` z polem `name`. Dodaj do klasy metodę `introduceYourself`, która wyświetli wiadomość "Hi, I'm" i imię osoby. W klasie `TestPerson`, utwórz obiekt `Person` i wywołaj na nim metodę `introduceYourself`. Czy musisz użyć słowo kluczowe `this` w implementacji metody?

    B. Dodaj do klasy `Person` metodę `sayHello`, która jako argument przyjmuje inny obiekt klasy `Person` i wyświetla wiadomość "Hello," i imię drugiej osoby. Przeanalizuj działanie.

    C. Dodaj do klasy `Person` metodę `changeName`, która jako argument przyjmuje łańcuch znaków i przypisuje go do pola `name`. Utwórz obiekt `Person` i użyj metody `changeName` do zmiany jego imienia. Następnie wywołaj metodę `introduceYourself`. Czy imię zostało zmienione? Czy musisz użyć słowo kluczowe `this` w implementacji metody?

    D. Dodaj do klasy `Person` metodę `swapNames`, która jako argument przyjmuje inny obiekt klasy `Person` i zamienia imionami obie osoby. Utwórz dwa obiekty `Person` i użyj metody `swapNames` do zamiany ich imion. Następnie wywołaj metodę `introduceYourself` na obu obiektach. Czy imiona zostały zamienione?

13. Wykonaj kolejno poniższe czynności:

    A. Stwórz klasę `Counter` z jednym polem `number`. Dodaj do klasy metodę `increase`, która przyjmuje parametr typu int i zwiększa wartość pola `number` o wartość tego parametru. W klasie `TestCounter`, utwórz zmienną typu int, przekaż ją do metody `increase` i sprawdź, czy wartość zmiennej zmieniła się po wywołaniu metody.

B. Dodaj do klasy `Counter` metodę `add`, która przyjmuje inny obiekt `Counter` i dodaje wartość jego pola `number` do pola `number` bieżącego obiektu. Utwórz dwa obiekty `Counter` w klasie `TestCounter` i użyj metody `add`. Sprawdź, czy obiekt przekazany jako argument metody `add` zmienił swoją wartość po wywołaniu metody.

C. Stwórz klasę `Modifier`, która posiada metodę `changeValue`, która jako argument przyjmuje typ int. Wewnątrz metody zmień wartość argumentu na inną. W klasie `TestCounter` utwórz zmienną int, a następnie przekaż ją do metody `changeValue` i sprawdź, czy wartość zmiennej się zmieniła.

D. Dodaj do klasy `Modifier` metodę `changeObject`, która jako argument przyjmuje obiekt `Counter`. Wewnątrz metody zmień wartość pola `number` obiektu `Counter` na inną. W klasie `TestCounter` utwórz obiekt `Counter`, a następnie przekaż go do metody `changeObject` i sprawdź, czy wartość pola `number` obiektu się zmieniła.

10 Modyfikatory dostępu

1. Utwórz klasę `Person` z publicznym polem `name` oraz prywatnym polem `password`. Zobacz jak różne modyfikatory dostępu wpływają na dostęp do tych pól z innej klasy.
2. Stwórz dwie klasy: `Parent` i `Child`. Klasa `Parent` powinna mieć jedno pole `protected`. Spróbuj uzyskać dostęp do tego pola z klasy `Child`.
3. Utwórz klasę `Car` z prywatną metodą `engineFailure()`. Spróbuj wywołać tę metodę z zewnątrz klasy.
4. Stwórz dwie klasy w tym samym pakiecie: `Employee` i `Company`. Klasa `Employee` powinna mieć pole bez modyfikatora dostępu. Spróbuj uzyskać dostęp do tego pola z klasy `Company`.
5. Utwórz klasę `BankAccount` z publicznym polem `accountNumber` i prywatnym polem `balance`. Zobacz, jak różne modyfikatory dostępu wpływają na dostęp do tych pól z innej klasy.
6. Stwórz klasę `Player` z trzema polami: `name` (publiczne), `age` (pomijając modyfikator dostępu) oraz `nationality` (prywatne). W klasie `Main` stwórz tablicę zawodników i spróbuj zmienić wartości wszystkich pól. Obserwuj rezultaty.
7. Stwórz klasę `Coach`, która jest zadeklarowana jako `private`. Spróbuj stworzyć obiekt `Coach` w klasie `Main`. Jaki jest rezultat?

11 Konstruktor

1. Napisz klasę `Book`, która będzie zawierać trzy pola: `title`, `author`, `publicationYear`. Następnie zaimplementuj dwa konstruktory - jeden domyślny, który nie przyjmuje żadnych argumentów, i drugi, który przyjmuje trzy argumenty odpowiadające polom klasy. W przypadku drugiego konstruktora, nazwy parametrów muszą być takie same jak nazwy pól. Sprawdź czy jesteś w stanie prawidłowo przypisać wartości do pól klasy używając słowa kluczowego `this`.

2. Zaprojektuj klasę `Person`, która będzie zawierać dwa pola: `firstName`, `lastName`. Zaimplementuj konstruktor, który przyjmuje dwa argumenty odpowiadające polom klasy. Nazwy parametrów muszą być takie same jak nazwy pól. Spróbuj przypisać wartości do pól klasy bez używania słowa kluczowego `this`. Czy jest to możliwe? Jeżeli nie, to dlaczego?

3. Stwórz klasę `Car`, która będzie zawierać trzy pola: `brand`, `model`, `productionYear`. Zaimplementuj trzy konstruktory - pierwszy domyślny, drugi przyjmujący dwa argumenty (`brand` i `model`), trzeci przyjmujący trzy argumenty (`brand`, `model`, `productionYear`). W przypadku drugiego i trzeciego konstruktora, nazwy parametrów muszą być takie same jak nazwy pól. Wykorzystaj słowo kluczowe `this` do rozróżnienia pól klasy od parametrów.

4. Napisz klasę `Person`, która będzie miała pola: `firstName`, `lastName` i `age`. Zdefiniuj w niej dwa konstruktory, jeden przyjmujący wszystkie trzy parametry, a drugi tylko imię i nazwisko. Drugi konstruktor powinien wywołać pierwszy, przekazując mu domyślną wartość wieku jako 0.

5. Zaprojektuj klasę `Square`, która posiada pole `side` oraz konstruktor, który umożliwia ustawienie wartości tego pola. Następnie napisz drugi konstruktor, który nie przyjmuje żadnych argumentów, a jedynie wywołuje pierwszy konstruktor z wartością domyślną 1.

6. Zdefiniuj klasę `Car`, która ma pola: `brand`, `model` i `productionYear`. Klasa powinna zawierać dwa konstruktory: pierwszy przyjmujący wszystkie trzy parametry, a drugi tylko markę i model. Drugi konstruktor powinien wywoływać pierwszy, przekazując mu domyślną wartość `productionYear` jako aktualny rok.

7. Stwórz klasę `Dog`, która posiada pola: `name`, `breed` i `age`. Klasa powinna mieć dwa konstruktory: jeden, który przyjmuje wszystkie trzy parametry, a drugi, który przyjmuje tylko imię i rasę. Drugi konstruktor powinien wywoływać pierwszy, przekazując mu domyślną wartość wieku jako 1.

8. Napisz klasę `Student`, która ma pola: `firstName`, `lastName` i `fieldOfStudy`. Zdefiniuj w niej dwa konstruktory, jeden przyjmujący wszystkie trzy parametry, a drugi tylko `firstName` i `lastName`. Drugi konstruktor powinien wywoływać pierwszy, przekazując mu domyślną wartość `fieldOfStudy` jako `unknown`.

9. Zdefiniuj klasę `Employee`, która posiada pola: `firstName`, `lastName` i `salary`. Napisz konstruktor, który przyjmuje dwa argumenty (`firstName` i `lastName`), oraz blok inicjujący, który ustawia wartość `salary` na 3000. Sprawdź, co stanie się, gdy w konstruktorze spróbujesz nadpisać wartość `salary`.

10. Zaprojektuj klasę `Car`, która posiada pola: `brand`, `model` i `price`. Napisz konstruktor, który przyjmuje dwa argumenty (`brand` i `model`), a w bloku inicjującym ustaw domyślną wartość`price` na 50000. Zobacz, co stanie się, gdy w konstruktorze spróbujesz zmienić wartość `price`.

11. Zdefiniuj klasę `Phone`, która ma pola: `manufacturer`, `model` i `operatingSystem`. W bloku inicjującym ustaw wartość `operatingSystem` na `Android`. Następnie napisz konstruktor, który przyjmuje dwa argumenty (`manufacturer` i `model`) i próbuje nadpisać wartość operatingSystem na `iOS`. Sprawdź, która wartość zostanie ostatecznie przypisana do pola operatingSystem.

12. Stwórz klasę `Car` zawierającą prywatne pola: `brand`, `model`, `productionYear`, `mileage` oraz `color`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) dla wszystkich pól. Następnie dodaj metodę `displayInformation()`, która wyświetla wszystkie informacje o samochodzie.

13. Stwórz klasę `Person` z prywatnymi polami: `firstName`, `lastName`, `age`, `address`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `introduceYourself()`, która zwraca łańcuch znaków z informacjami o osobie.

14. Stwórz klasę `Book` z prywatnymi polami: `title`, `author`, `publicationYear`, `publisher` oraz `numberOfPages`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `showInformation()`, która wyświetla informacje o książce.

15. Stwórz klasę `Point2D` z prywatnymi polami `x` i `y`, reprezentującymi współrzędne punktu na płaszczyźnie. Dodaj konstruktor, który przyjmuje współrzędne jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `distance(Point2D anotherPoint)`, która oblicza odległość między dwoma punktami na płaszczyźnie.

16. Stwórz klasę `Rectangle` z prywatnymi polami `width` i `height`. Dodaj konstruktor, który przyjmuje długości boków jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `area()` i `perimeter()`, które obliczają pole powierzchni i obwód prostokąta.

17. Stwórz klasę `Circle` z prywatnym polem `radius`. Dodaj konstruktor, który przyjmuje promień jako argument. Dodaj metody dostępowe (gettery i settery) oraz metody `area()` i `circumference()`, które obliczają pole powierzchni i obwód koła.

18. Stwórz klasę `Student` z prywatnymi polami: `firstName`, `lastName`, `indexNumber`, `yearOfStudy` oraz `gradeAverage`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `showInformation()`, która wyświetla informacje o studencie.

19. Stwórz klasę `Employee` z prywatnymi polami: `firstName`, `lastName`, `position`, `age` oraz `salary`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `showInformation()`, która wyświetla informacje o pracowniku.

20. Stwórz klasę `BankAccount` z prywatnymi polami: `accountNumber`, `owner`, `balance` oraz `accountType`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `deposit(double amount)` i `withdraw(double amount)`, które odpowiednio dodają lub odejmują kwotę od salda konta.

21. Stwórz klasę `Television` z prywatnymi polami: `brand`, `screenDiagonal`, `resolution`, `isSmartTV` oraz `price`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `showInformation()`, która wyświetla informacje o telewizorze.

22. Stwórz klasę `Gradebook` z prywatnymi polami: `firstName`, `lastName` oraz `grades` (jako `ArrayList` typu `int`). Dodaj konstruktor, który przyjmuje firstName i lastName jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `addGrade(int grade)` i `removeGrade(int index)`, które odpowiednio dodają lub usuwają ocenę z listy ocen. Dodaj również metodę `averageGrade()` do obliczania i zwracania średniej ocen.

23. Stwórz klasę `TemperatureHistory` z prywatnym polem `temperatures` (jako `ArrayList` typu `double`). Dodaj konstruktor domyślny. Dodaj metody dostępowe (gettery i settery) oraz metody `addTemperature(double temperature)` i `removeTemperature(int index)`, które odpowiednio dodają lub usuwają temperaturę z listy temperatur. Dodaj również metodę `averageTemperature()` do obliczania i zwracania średniej temperatur.

24. Stwórz klasę `Results` (wyniki testu) z prywatnymi polami: `firstName`, `lastName` oraz `results` (jako tablica typu `int`). Dodaj konstruktor, który przyjmuje `firstName`, `lastName` oraz rozmiar tablicy jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `addResult(int index, int result)`, która dodaje wynik testu na podanym indeksie. Dodaj również metodę `averageResult()` do obliczania i zwracania średniego wyniku.

25. Stwórz klasę `TaskManager` z prywatnym polem `taskPriorities` (jako `ArrayList` typu `int`). Dodaj konstruktor domyślny. Dodaj metody dostępowe (gettery i settery) oraz metody `addPriority(int priority)` i `removePriority(int index)`, które

odpowiednio dodają lub usuwają priorytet z listy priorytetów. Dodaj również metodę `highestPriority()` do znajdowania i zwracania wartości najwyższego priorytetu.

26. Stwórz klasę `Warehouse` z prywatnym polem `productQuantities` (jako tablica typu `int`). Dodaj konstruktor, który przyjmuje rozmiar tablicy jako argument. Dodaj metody dostępowe (getters and setters) oraz metodę `addProducts(int index, int quantity)`, która dodaje określoną ilość produktów na podanym indeksie. Dodaj również metodę `totalProducts()` do obliczania i zwracania sumy wszystkich produktów w magazynie.

27. Zdefiniuj klasę `Person`, która posiada pola: `firstName`, `lastName` i `age`. Napisz konstruktor, który przyjmuje trzy argumenty i waliduje je przed przypisaniem do odpowiednich pól. Wiek osoby (`age`) nie powinien być ujemny, a `firstName` i `lastName` nie powinny być puste i nullem (w przypadku niepoprawnych wartości z osobna ustaw pusty napis lub zero).

28. Stwórz klasę `Point`, reprezentującą punkt w przestrzeni 3D, z polami: `x`, `y`, `z`. Napisz konstruktor, który przyjmuje te trzy wartości i sprawdza, czy są one w zakresie od -100 do 100. Jeśli wartości nie są w tym zakresie, powinny być ustawione na najbliższą granicę.

29. Zaprojektuj klasę `Car`, która posiada pola: `brand`, `model` i `productionYear`. Napisz konstruktor, który przyjmuje trzy argumenty. Zadaniem konstruktora jest sprawdzenie, czy `productionYear` nie jest większy niż aktualny rok oraz czy `brand` i `model` nie są puste i nullem (w przypadku niepoprawnych wartości z osobna ustaw pusty napis lub bieżący rok).

Wskazówka: wykorzystaj `Calendar.getInstance().get(Calendar.YEAR)` do pobrania aktualnego roku.

30. Zdefiniuj klasę `BankAccount`, która posiada pola: `accountNumber` i `balance`. Napisz konstruktor, który przyjmuje dwa argumenty. Przed przypisaniem wartości do pola `accountNumber`, sprawdź, czy jest ono 26-cyfrowe, a przed przypisaniem wartości do pola `balance`, sprawdź, czy `balance` nie jest ujemne (w przypadku niepoprawności wartości ustaw numer konta na taki, by składał się z samych jedynek a saldo na zero).

12 Pola i metody statyczne

1. Stwórz klasę `Person`. Klasa powinna zawierać pole `name` oraz statyczne pole `counter`. Zadaniem jest zwiększanie wartości `counter` za każdym razem, gdy tworzona jest nowa instancja klasy `Person`. Dodaj metodę, która zwróci wartość `counter`. Stwórz przypadek testowy.

2. Stwórz klasę `Mathematics`, która posiada pole statyczne `PI`, które przechowuje przybliżoną wartość liczby π. Upewnij się, że wartość tego pola jest niemodyfikowalna. Stwórz przypadek testowy.

3. Stwórz klasę `Singleton`, która używa pola statycznego do przechowywania jednej i tylko jednej instancji tej klasy. Klasa powinna zawierać prywatny konstruktor oraz publiczną statyczną metodę `getInstance()`, która zwraca jedyną instancję klasy `Singleton`. Stwórz przypadek testowy.

4. Stwórz klasę `Configuration`, która zawiera pole statyczne `applicationVersion` z domyślną wartością "1.0". Dodaj metody pozwalające na odczyt i zmianę tej wartości. Stwórz przypadek testowy.

5. Stwórz klasę `Calculator`, która posiada podstawowe metody matematyczne: dodawanie, odejmowanie, mnożenie i dzielenie. Zaimplementuj te metody jako metody statyczne. Utwórz drugą klasę, w której przetestujesz te metody bez tworzenia instancji klasy `Calculator`.

6. Stwórz klasę `ShapeFactory`, która posiada metody statyczne do tworzenia różnych figur geometrycznych, takich jak koła, kwadraty czy trójkąty. Następnie stwórz klasę testową, w której wykorzystasz te metody do stworzenia różnych figur, bez potrzeby tworzenia instancji klasy `ShapeFactory`.

7. Stwórz klasę `Settings`, która posiada różne parametry konfiguracyjne aplikacji (np. wersja, język). Dodaj metody niestatyczne do zmiany tych parametrów oraz statyczną metodę `defaultSettings()`, która zwróci domyślną konfigurację aplikacji. W klasie testowej zobacz, jak działają obie grupy metod i jakie są między nimi różnice.

8. Utwórz klasę `Configuration`, która posiada pole statyczne `applicationVersion` oraz pole statyczne `applicationName`. Jedno z tych pól oznacz jako `final`. Zastanów się, co by się stało, gdybyś chciał zaktualizować wartość jednego z tych pól w trakcie działania aplikacji.

9. Stwórz klasę `Product`, która zawiera pole statyczne `numberOfProducts` oraz pole statyczne `MAX_PRODUCTS`. Pole `numberOfProducts` będzie służyć do zliczania ilości utworzonych produktów, a `MAX_PRODUCTS` do ograniczenia ich liczby. Oznacz tylko jedno z tych pól słowem kluczowym `final` i zastanów się nad konsekwencjami tego wyboru.

10. W jednym projekcie wykonaj czynności:

A. Stwórz klasę `SportsFacility`, która powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.sports`.

B. Klasa powinna zawierać trzy atrybuty:

- `type` (typ obiektu), typu `String`.
- `location`, typu `String`.
- `capacity` (pojemność), typu `int` (w liczbie osób).

C. W klasie `SportsFacility`, zaimplementuj statyczną metodę `createStadium(String type, String location, int capacity)`. Metoda ma zwrócić nowy obiekt typu `SportsFacility`, którego pola ustawione są z argumentów metody.

D. W klasie `SportsFacility`, zaimplementuj nie-statyczną metodę `createFacility(String type, String location, int capacity)`. Metoda ma zwrócić nowy obiekt typu `SportsFacility`, którego pola ustawione są z argumentów metody.

E. Stwórz klasę `TestSportsFacility`, umieść ją w innym pliku w pakiecie `pl.edu.uwm.wmii.sports`. W klasie `TestSportsFacility` dodaj metodę `main`. Wywołaj w niej metody z punktu C i D.

11. W jednym projekcie wykonaj czynności:

A. Stwórz klasę `Software`, która powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.software`.

B. Klasa powinna zawierać trzy atrybuty:

- `type` (typ oprogramowania), typu `String`.
- `version`, typu `String`.
- `licenseCount` (liczba licencji), typu `int`.

C. W klasie `Software`, zaimplementuj statyczną metodę `isLicenseAvailable` z argumentem typu `Software`. Metoda ma zwrócić wartość logiczną sprawdzającą czy liczba licencji oprogramowania przekazanego jako argument jest większa niż 3. Dla argumentu będącego `null`em zwróć `false`.

D. W klasie `Software`, zaimplementuj nie-statyczną metodę `isLicenseAvailable` bez argumentu. Metoda ma zwrócić wartość logiczną sprawdzającą czy liczba licencji bieżącego obiektu jest większa niż 3.

E. Stwórz klasę `TestSoftware`, umieść ją w innym pliku w pakiecie `pl.edu.uwm.wmii.software`. W klasie `TestSoftware` dodaj metodę `main`. Wywołaj w niej metody z punktu C i D.

13 Rekordy

1. Stwórz rekord `BookDTO`, który reprezentuje książkę w sklepie internetowym. Powinien zawierać takie informacje jak `title`, `author`, `price` i `yearOfPublication`. Następnie stwórz kilka instancji tego rekordu, reprezentujących różne książki.
2. Utwórz rekord `Address`, który zawiera `street`, `houseNumber`, `postalCode` i `city`. Następnie, stwórz rekord `Person`, który oprócz podstawowych informacji o osobie (np. `firstName`, `lastName`) zawiera również pole typu `Address`. Stwórz przypadek testowy.
3. Stwórz klasę `PointC` reprezentującą punkt na płaszczyźnie 2D z polami `x` i `y`. Następnie, stwórz rekord `PointR` z tymi samymi polami. Dodaj metody umożliwiające konwersję z instancji klasy na rekord i odwrotnie. Stwórz przypadek testowy.
4. Stwórz rekord `Book`, który reprezentuje `title`, `author` i `yearOfPublication`. Dodaj metodę `describe()`, która zwraca sformatowaną postać informacji o książce w stylu "Autor - Tytuł (Rok wydania)". Stwórz przypadek testowy.
5. Stwórz rekord `Point2D`, reprezentujący punkt na płaszczyźnie 2D z koordynatami `x` i `y`. Wprowadź metodę `distanceTo(Point2D otherPoint)`, która oblicza i zwraca odległość euklidesową między aktualnym punktem a podanym jako argument. Stwórz przypadek testowy.
6. Stwórz rekord `Car`, który zawiera `brand`, `model` i `fuelConsumptionPer100km`. Dodaj metodę `fuelCost(double fuelPrice, double distance)`, która oblicza i zwraca koszt podróży na podstawie podanej ceny paliwa oraz dystansu. Stwórz przypadek testowy.
7. Stwórz rekord `Student`, który reprezentuje imię, nazwisko i listę ocen ucznia. Wprowadź metodę `averageGrades()`, która oblicza średnią ocen ucznia. Stwórz przypadek testowy.
8. Stwórz rekord `Order`, który zawiera listę produktów oraz ich cen. Każdy produkt reprezentowany jest przez rekord `Product` z nazwą i ceną. W rekordzie `Order` dodaj metodę `totalValue()`, która oblicza łączną wartość zamówienia na podstawie cen produktów. Stwórz przypadek testowy.
9. Stwórz rekord `Person`, który reprezentuje imię i wiek osoby. Dodaj konstruktor, który weryfikuje, czy podany wiek nie jest ujemny. W przypadku podania wartości ujemnej, ustaw wiek na 0. Stwórz przypadek testowy.

10. Stwórz rekord `BankAccount`, który zawiera numer konta oraz saldo. Dodaj konstruktor, który pozwala na tworzenie konta tylko z numerem, przy czym domyślne saldo wynosi 0. Stwórz przypadek testowy.

11. Stwórz rekord `MusicTrack`, który opisuje tytuł, artystę i czas trwania utworu w sekundach. Dodaj konstruktor, który przyjmuje tylko tytuł i artystę, przyjmując domyślną długość utworu jako 180 sekund. Stwórz przypadek testowy.

14 Metody `toString`, `equals` i `hashCode`.

1. Wykonaj poniższe czynności:

- Zdefiniuj klasę `Person`, która posiada następujące pola: `firstName`, `lastName` i `age`.
- Napisz konstruktor, który przyjmuje trzy argumenty i waliduje je przed przypisaniem do odpowiednich pól.
  - Wiek osoby (`age`) nie powinien być ujemny. W przypadku podania wartości ujemnej dla wieku, ustaw wiek osoby na zero.
  - Pola `firstName` i `lastName` nie powinny być puste ani równać się null. W przypadku podania pustego napisu lub null dla tych pól, ustaw odpowiednio pusty napis.
- Dodaj metodę `toString()`, która zwraca informacje o osobie w formacie: `"Person: [firstName] [lastName], Age: [age]."`. Zwróć uwagę na wielkość liter i znaki interpunkcyjne.
- Dodaj metodę `equals()`, która porównuje dwie osoby na podstawie ich pól `firstName`, `lastName` i `age`. Dwie osoby są uważane za identyczne, jeśli wszystkie trzy pola są takie same.
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

2. Wykonaj poniższe czynności:

- Zdefiniuj klasę `Vehicle`, która posiada następujące pola: `brand`, `model` i `yearOfProduction`.
- Napisz konstruktor, który przyjmuje trzy argumenty i waliduje je przed przypisaniem do odpowiednich pól.
  - Rok produkcji (`yearOfProduction`) nie powinien być większy od aktualnego roku. W przypadku podania wartości większej, ustaw rok produkcji na aktualny rok.
  - Pola `brand` i `model` nie powinny być puste ani równać się null. W przypadku podania pustego napisu lub null dla tych pól, ustaw odpowiednio pusty napis.
- Dodaj metodę `toString()`, która zwraca informacje o pojeździe w formacie: `"Vehicle: [brand] [model], Year: [yearOfProduction]."`.
- Dodaj metodę `equals()`, która porównuje dwa pojazdy na podstawie ich pól `brand`, `model` i `yearOfProduction`.

- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

3. Wykonaj poniższe czynności:

- Zdefiniuj klasę `Book`, która posiada następujące pola: `title`, `author` i `numberOfPages`.
- Napisz konstruktor, który przyjmuje trzy argumenty i waliduje je przed przypisaniem do odpowiednich pól.
  - Liczba stron (`numberOfPages`) nie powinna być ujemna. W przypadku podania wartości ujemnej, ustaw liczbę stron na jedną.
  - Pola `title` i `author` nie powinny być puste ani równać się null. W przypadku podania pustego napisu lub null, ustaw odpowiednio pusty napis.
- Dodaj metodę `toString()`, która zwraca informacje o książce w formacie: `"Book: [title] by [author], Pages: [numberOfPages]."`.
- Dodaj metodę `equals()`, która porównuje dwie książki na podstawie ich pól `title`, `author` i `numberOfPages`.
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

4. Wykonaj poniższe czynności:

- Zdefiniuj klasę `Product`, która posiada następujące pola: `productName`, `category` i `price`.
- Napisz konstruktor, który przyjmuje trzy argumenty i waliduje je przed przypisaniem do odpowiednich pól.
  - Cena (`price`) nie powinna być ujemna. W przypadku podania wartości ujemnej, ustaw cenę produktu na zero.
  - Pola `productName` i `category` nie powinny być puste ani równać się null. W przypadku podania pustego napisu lub null, ustaw odpowiednio pusty napis.
- Dodaj metodę `toString()`, która zwraca informacje o produkcie w formacie: `"Product: [productName], Category: [category], Price: [price]."`.
- Dodaj metodę `equals()`, która porównuje dwa produkty na podstawie ich pól `productName`, `category` i `price`.
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

4. Wykonaj poniższe czynności:

- Stwórz klasę `Results` (wyniki testu) z prywatnymi polami: `firstName`, `lastName` oraz `results` (jako tablica typu `int`). Dodaj konstruktor, który przyjmuje `firstName`, `lastName` oraz rozmiar tablicy jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `addResult(int index, int result)`, która dodaje wynik testu na podanym indeksie. Dodaj również metodę `averageResult()` do obliczania i zwracania

średniego wyniku.

- Dodaj metodę `toString()`, która zwraca informacje o uczniu, jego wynikach oraz zawartość tablicy `results` w formacie: `"Results for [firstName] [lastName]: Average Score = [averageResult], Results: [result1, result2, ...]. "`. Ponadto dodaj metodę `equals()`, która porównuje dwa obiekty klasy `Results` na podstawie ich pól `firstName`, `lastName` oraz zawartości tablicy `results`. Dwa obiekty są uważane za identyczne, jeśli wszystkie pola i wyniki testów w tablicach są takie same. Dodaj także metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

5. Wykonaj poniższe czynności:

- Stwórz klasę `CDAlbum` z prywatnymi polami: `artistName`, `albumTitle` oraz `tracks` (jako tablica typu `int` reprezentująca długość każdego utworu w minutach).
- Dodaj konstruktor, który przyjmuje `artistName`, `albumTitle` oraz ilość utworów jako argumenty.
- Dodaj metody dostępowe (gettery i settery) oraz metodę `addTrack(int index, int length)`, która dodaje długość utworu na podanym indeksie.
- Dodaj również metodę `totalLength()` do obliczania i zwracania łącznej długości wszystkich utworów.
- Dodaj metodę `toString()`, która zwraca informacje o artyście, albumie oraz czasie trwania wszystkich utworów w formacie: `"CDAlbum by [artistName]: [albumTitle], Total Length = [totalLength] minutes, Tracks Length: [length1, length2, ...]. "`.
- Dodaj metodę `equals()`, która porównuje dwa obiekty klasy `CDAlbum` na podstawie ich pól `artistName`, `albumTitle` oraz zawartości tablicy `tracks`. Dwa albumy są uważane za identyczne, jeśli wszystkie informacje są takie same.
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

6. Wykonaj poniższe czynności:

- Stwórz klasę `MovieRating` z prywatnymi polami: `movieTitle`, `directorName` oraz `ratings` (jako tablica typu `double` reprezentująca oceny filmu w skali od 1 do 10).
- Dodaj konstruktor, który przyjmuje `movieTitle`, `directorName` oraz ilość ocen jako argumenty.
- Dodaj metody dostępowe (gettery i settery) oraz metodę `addRating(int index, double rating)`, która dodaje ocenę filmu na podanym indeksie.
- Dodaj również metodę `averageRating()` do obliczania i zwracania średniej oceny filmu.
- Dodaj metodę `toString()`, która zwraca informacje o filmie, reżyserze, średniej ocenie oraz wszystkich ocenach w formacie: `"MovieRating for [movieTitle] by [directorName]: Average Rating = [averageRating], Ratings: [rating1, rating2, ...]. "`.

- Dodaj metodę `equals()`, która porównuje dwa obiekty klasy `MovieRating` na podstawie ich pól `movieTitle`, `directorName` oraz zawartości tablicy `ratings`. Dwa filmy są uważane za identyczne, jeśli wszystkie informacje są takie same.
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

7. Wykonaj poniższe czynności:

- Stwórz klasę `Gradebook` z prywatnymi polami: `firstName`, `lastName` oraz `grades` (jako `ArrayList` typu `int`). Dodaj konstruktor, który przyjmuje firstName i lastName jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `addGrade(int grade)` i `removeGrade(int index)`, które odpowiednio dodają lub usuwają ocenę z listy ocen. Dodaj również metodę `averageGrade()` do obliczania i zwracania średniej ocen.
- Dodaj metodę `toString()`, która zwraca informacje o uczniu, średniej jego ocen oraz wszystkich ocenach w formacie: `"Gradebook for [firstName] [lastName]: Average Grade = [averageGrade], Grades: [grade1, grade2, ...]."`. Zwróć uwagę na wielkość liter i znaki interpunkcyjne.
- Dodaj metodę `equals()`, która porównuje dwa obiekty klasy `Gradebook` na podstawie ich pól `firstName`, `lastName` oraz zawartości listy `grades`. Dwa dzienniczki są uważane za identyczne, jeśli mają takie same imię, nazwisko i identyczny zestaw ocen (z uwzględnieniem kolejności).
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

8. Wykonaj poniższe czynności:

- Stwórz klasę `Wallet` z prywatnymi polami: `ownerName`, `ownerSurname` oraz `coins` (jako `ArrayList` typu `double`). Dodaj konstruktor, który przyjmuje `ownerName` i `ownerSurname` jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `addCoin(double coin)` i `removeCoin(int index)`, które odpowiednio dodają lub usuwają monetę z listy monet.
- Dodaj również metodę `totalAmount()` do obliczania i zwracania łącznej kwoty w portfelu.
- Dodaj metodę `toString()`, która zwraca informacje o właścicielu portfela, łącznej kwocie oraz wszystkich monetach w formacie: `"Wallet of [ownerName] [ownerSurname]: Total Amount = [totalAmount], Coins: [coin1, coin2, ...]."`. Zwróć uwagę na wielkość liter i znaki interpunkcyjne.
- Dodaj metodę `equals()`, która porównuje dwa obiekty klasy `Wallet` na podstawie ich pól `ownerName`, `ownerSurname` oraz zawartości listy `coins`. Dwa portfele są uważane za identyczne, jeśli mają tego samego właściciela i identyczny zestaw monet (bez uwzględnienia kolejności).
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

9. Wykonaj poniższe czynności:

- Stwórz klasę `LibraryCard` z prywatnymi polami: `holderFirstName`, `holderLastName` oraz `booksBorrowed` (jako `ArrayList` typu `String`). Dodaj konstruktor, który przyjmuje `holderFirstName` i `holderLastName` jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `borrowBook(String bookTitle)` i `returnBook(String bookTitle)`, które odpowiednio dodają lub usuwają tytuł książki z listy wypożyczonych książek.
- Dodaj również metodę `numberOfBooksBorrowed()` do obliczania i zwracania liczby wypożyczonych książek.
- Dodaj metodę `toString()`, która zwraca informacje o posiadaczu karty, liczbie wypożyczonych książek oraz wszystkich tytułach wypożyczonych książek w formacie: `"Library card of [holderFirstName] [holderLastName]: Number of books borrowed = [numberOfBooksBorrowed], Books: [book1, book2, ...]."`. Zwróć uwagę na wielkość liter i znaki interpunkcyjne.
- Dodaj metodę `equals()`, która porównuje dwa obiekty klasy `LibraryCard` na podstawie ich pól `holderFirstName`, `holderLastName` oraz zawartości listy `booksBorrowed`. Dwóch posiadaczy kart bibliotecznych uważa się za identycznych, jeśli mają takie same imię, nazwisko i identyczny zestaw wypożyczonych książek (z zachowaniem kolejności).
- Dodaj metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`

15 Dziedziczenie

1. Wykonaj kolejno poniższe czynności:

   A. Stwórz klasę bazową `Person` z prywatnym polem `firstName` oraz chronionym polem `lastName`. Następnie stwórz klasę `Employee`, która dziedziczy po klasie `Person`. W klasie `Employee` próbuj odnieść się do obu pól i zauważ, które z nich są dostępne.
   B. Na bazie klasy `Person` z poprzedniego podpunktu, stwórz metody dostępowe (gettery) dla obu pól. W klasie `Employee` stwórz metodę `displayData`, która korzysta z tych metod dostępowych, aby wypisać informacje o pracowniku. Zastanów się, dlaczego metody dostępowe są używane do dostępu do prywatnych pól.

2. Stwórz klasę bazową `Book` z prywatnym polem `title` oraz chronionym polem `author`. Stwórz klasę potomną `Ebook`, która dziedziczy po klasie `Book`. W klasie `Ebook` spróbuj zmienić modyfikator dostępu dla obu pól z klasy bazowej. Zastanów się, dlaczego jedno z pól pozwala na to, a drugie nie.

3. Stwórz klasę `Tool` z chronionym konstruktorem, który przyjmuje nazwę narzędzia. Następnie stwórz klasę potomną `Hammer`, która dziedziczy po klasie `Tool`. W klasie `Hammer` stwórz konstruktor, który korzysta z konstruktora klasy bazowej. Zastanów się, dlaczego używając modyfikatora `private` dla konstruktora klasy bazowej, taki scenariusz nie byłby możliwy.

4. Stwórz dwa różne pakiety: `animals` i `mammals`. W pakiecie `animals` stwórz klasę bazową `Animal` z chronionym polem `species` i prywatnym polem `age`. W pakiecie `mammals` stwórz klasę `Dog`, która dziedziczy po klasie `Animal`. Spróbuj odnieść się w klasie `Dog` do obu pól z klasy bazowej i zauważ, które z nich są dostępne.

5. Wykonaj kolejno poniższe czynności:

   A. Stwórz klasę bazową `Vehicle` z metodą `drive`, która wypisuje "The vehicle is moving.". Następnie stwórz klasę potomną `Car`, która nadopisuje metodę `drive` tak, by wypisywała "The car is moving.". Utwórz obiekt klasy `Car` i wywołaj jego metodę `drive`, aby zaobserwować wynik.
   B. Na bazie klasy `Vehicle` z poprzedniego podpunktu, w klasie `Car`, nadpisz metodę `drive` tak, by wywoływała oryginalną metodę z klasy bazowej i dodatkowo wypisywała informacje specyficzne dla klasy `Car`. Użyj słowa kluczowego `super`, aby wywołać metodę z klasy bazowej.

6. Stwórz klasę `Calculator` z metodą `add`, która przyjmuje dwa argumenty typu `int` i zwraca ich sumę. Następnie stwórz klasę potomną `ExtendedCalculator`, która nadpisuje metodę `add` w taki sposób, by przyjmowała trzy argumenty typu `int` i zwracała ich sumę. Zastanów się, czy to faktycznie nadpisywanie metody, czy może coś innego.

7. Stwórz klasę bazową `Base` z metodą statyczną `info`, która wypisuje "This is the base class.". Następnie stwórz klasę potomną `Child` i próbuj przesłonić metodę statyczną `info` tak, by wypisywała "This is the child class.". Zastanów się nad zachowaniem tak przesłoniętych metod i dlaczego takie przesłanianie jest inaczej traktowane.

8. Stwórz klasę bazową `Computer` z chronioną metodą `start`, która wypisuje "Computer started.". Stwórz klasę potomną `Laptop`, która próbuje nadpisać metodę `start`, ale z modyfikatorem dostępu `public`. Spróbuj skompilować kod i zastanów się, dlaczego występują pewne ograniczenia w nadpisywaniu metod pod względem modyfikatorów dostępu.

9. Stwórz klasę bazową o nazwie `Vehicle` z polami: `brand` i `model`. Klasa ta powinna posiadać konstruktor przyjmujący oba te parametry. Następnie stwórz klasę potomną o nazwie `Car`, która dziedziczy po klasie `Vehicle`. Klasa `Car` powinna posiadać dodatkowe pole `numberOfDoors`. Stwórz konstruktor dla klasy `Car`, który przyjmuje wszystkie trzy parametry i korzysta z konstruktora klasy bazowej. Stwórz przypadek testowy.

10. Stwórz klasę bazową o nazwie `Animal` z polem `name` oraz konstruktor z domyślnym przypisaniem nazwy do "Unknown". Następnie stwórz klasę `Dog`, która dziedziczy po klasie `Animal`. Nie twórz w niej dodatkowego konstruktora. Sprawdź, jakie zwierzę zostanie utworzone, gdy stworzysz nową instancję klasy `Dog`.

11. Stwórz klasę bazową o nazwie `Building` z polami: `height` i `color`. Klasa ta powinna posiadać dwa konstruktory: jeden przyjmujący oba parametry i drugi bezparametrowy, który przypisuje domyślne wartości. Stwórz klasę potomną `House` z dodatkowym polem `numberOfRooms`. Klasa `House` powinna posiadać konstruktor, który wykorzystuje przeciążony konstruktor klasy bazowej. Stwórz przypadek testowy.

12. Stwórz trzy klasy: `Entity`, `Human` i `Programmer`. Klasa `Human` powinna dziedziczyć po klasie `Entity`, a klasa `Programmer` po klasie `Human`. W każdej z klas dodaj konstruktor, który wypisuje informację o tworzeniu instancji danej klasy. Stwórz instancję klasy `Programmer` i zaobserwuj kolejność wywoływania konstruktorów.

13. Utwórz klasę `Vehicle` z polami `brand`, `model` i `yearOfProduction`. Utwórz klasy `Car` i `Motorcycle`, które dziedziczą po klasie `Vehicle`. Klasa `Car` powinna mieć dodatkowe pole `numberOfDoors`, a klasa `Motorcycle` pole `engineCapacity`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

14. Utwórz klasę `Employee` z polami `firstName`, `lastName` i `salary`. Utwórz klasy `Programmer` i `Tester`, które dziedziczą po klasie `Employee`. Klasa `Programmer` powinna mieć dodatkowe pole `programmingLanguage`, a klasa `Tester` pole `testingType`. Dodaj

konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

15. Utwórz klasę `Property` z polami `address`, `size` i `price`. Utwórz klasy `House` i `Apartment`, które dziedziczą po klasie `Property`. Klasa `House` powinna mieć dodatkowe pole `numberOfFloors`, a klasa `Apartment` pole `floorNumber`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

16. Utwórz klasę `BoardGame` z polami `gameName`, `minPlayers`, `maxPlayers` oraz `rules` (jako ArrayList typu `String`). Utwórz klasy `EducationalGame` i `StrategicGame`, które dziedziczą po klasie `BoardGame`. Klasa `EducationalGame` powinna mieć dodatkowe pole `subject`, a klasa `StrategicGame` pole `duration`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

17. Utwórz klasę `Team` z polami `name`, `city` oraz `points` (jako ArrayList typu `Integer`). Utwórz klasy `SoccerTeam` i `VolleyballTeam`, które dziedziczą po klasie `Team`. Klasa `SoccerTeam` powinna mieć dodatkowe pole `rankingPosition`, a klasa `VolleyballTeam` pole `numberOfVictories`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

18. Utwórz klasę `Computer` z polami `manufacturer`, `model` oraz `partsPrices` (jako ArrayList typu `Double`). Utwórz klasy `Laptop` i `Desktop`, które dziedziczą po klasie `Computer`. Klasa `Laptop` powinna mieć dodatkowe pole `weight`, a klasa `Desktop` pole `caseType`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

19. Utwórz klasę `MusicAlbum` z polami `title`, `artist` oraz `ratings` (jako ArrayList typu `Integer`). Utwórz klasy `RockAlbum` i `JazzAlbum`, które dziedziczą po klasie `MusicAlbum`. Klasa `RockAlbum` powinna mieć dodatkowe pole `rockGenre`, a klasa `JazzAlbum` pole `jazzGenre`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

20. Utwórz klasę `Book` z polami `author`, `title` i `yearOfPublication`. Następnie utwórz dwie klasy pochodne: `Ebook`, która dodaje pole `fileSize` i `PaperbackBook`, która dodaje pole `numberOfPages`. Każda z klas powinna zawierać konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()`. W metodzie `equals()` klasy `Book` porównaj tylko `author`, w klasie `Ebook` po `author` i `fileSize`, a w klasie `PaperbackBook` porównaj `author` i `numberOfPages`. Napisz program testujący te klasy, demonstrujący działanie polimorfizmu i porównanie obiektów.

21. Zaprojektuj klasę `Employee` z polami `name`, `department` i `salary`. Utwórz dwie klasy pochodne: `Manager` z dodatkowym polem `bonus` i `Intern` z polem `internshipLength`. Każda klasa powinna mieć konstruktory, gettery i settery, `toString()`, `equals()` oraz

`hashCode()`. W metodzie `equals()` klasy `Employee` porównaj tylko `name`, w klasie `Manager` po `name` i `bonus`, a w klasie `Intern` porównaj `name` i `internshipLength`. Napisz program testujący te klasy, demonstrujący działanie polimorfizmu i porównanie obiektów.

16 Pakiety

1. Stwórz pakiet o nazwie `cars`. Wewnątrz tego pakietu utwórz klasę `Car` zawierającą trzy prywatne pola: `brand`, `model` i `yearOfProduction`. Dodaj do klasy odpowiednie metody dostępowe (gettery i settery) oraz konstruktor przyjmujący wszystkie trzy pola. W klasie testującej, poza tym pakietem, stwórz obiekt klasy `Car`, nadaj mu wartości i wydrukuj je na ekran.

2. Stwórz pakiet o nazwie `animals`. Wewnątrz tego pakietu utwórz dwie klasy: `Dog` i `Cat`. Oba powinny zawierać pola `name` i `age`. Każda z klas powinna posiadać metodę `makeSound()`. Dla klasy `Dog` metoda ta powinna drukować "Woof, woof!", a dla klasy `Cat` - "Meow!". W klasie testującej, poza tym pakietem, stwórz obiekty obu klas, nadaj im wartości i wywołaj ich metody `makeSound()`.

3. Stwórz dwa pakiety: `employees` i `tools`. W pakiecie `employees` utwórz klasę `Employee` z polami `firstName`, `lastName` oraz `salary`. Pole `salary` powinno być chronione (protected). W pakiecie `tools` stwórz klasę `SalaryCalculator`, która posiada metodę `raiseSalary(Employee employee, double percent)`, która zwiększa pensję pracownika o podany procent. Spróbuj wywołać tę metodę w klasie testującej, poza oboma pakietami, i zastanów się nad problemami dostępu do chronionych pól w różnych pakietach.

4. Stwórz pakiet o nazwie `electronics`. Wewnątrz tego pakietu stwórz dwa podpakiety: `televisions` i `phones`. W podpakiecie `televisions` utwórz klasę `Television` z polami `brand` i `screenDiagonal`. W podpakiecie `phones` utwórz klasę `Phone` z polami `brand` i `operatingSystem`. Dla obu klas dodaj odpowiednie gettery, settery oraz konstruktory. W klasie testującej, poza pakietem `electronics`, stwórz obiekty obu klas, nadaj im wartości i wyświetl je.

5. Stwórz dwa pakiety: `books` i `library`. W pakiecie `books` stwórz klasę `Book` z polami `title`, `author` i `publicationYear`. W pakiecie `library` stwórz klasę `Shelf` zawierającą listę książek oraz metody umożliwiające umożliwiające dodawanie i usuwanie książek. Aby korzystać z klasy `Book` w pakiecie `library`, musisz zaimportować odpowiedni pakiet. W klasie testującej, stwórz kilka książek, dodaj je do półki i wydrukuj zawartość półki.

6. Stwórz pakiet `bank`. Wewnątrz tego pakietu stwórz dwie klasy: `Account` i `Bank`. Klasa `Account` powinna posiadać prywatne pola `accountNumber`, `balance` oraz metody `deposit(double amount)` i `withdraw(double amount)`. Klasa `Bank` powinna zawierać listę kont oraz metody do tworzenia new kont i realizacji przelewów między nimi. Spróbuj

utworzyć konto bezpośrednio w klasie testującej poza pakietem `bank` i zastanów się, jak modyfikatory dostępu wpłynęły na dostęp do klas i metod w pakiecie.

7. Stwórz pakiet o nazwie `animals`. Wewnątrz tego pakietu utwórz klasę bazową `Animal` z polami: `name` i `age` oraz metodą `makeSound()`. Następnie, w tym samym pakiecie, stwórz dwie klasy pochodne: `Dog` i `Cat`, które dziedziczą po klasie `Animal`. Klasy pochodne powinny nadpisywać metodę `makeSound()`. W klasie testującej, poza pakietem `animals`, stwórz obiekty obu klas pochodnych, nadaj im wartości i wywołaj metodę `makeSound()` dla każdego z nich.

8. Stwórz pakiet o nazwie `computers`. W tym pakiecie utwórz klasę bazową `Computer` z polami: `brand` i `processor`. Następnie stwórz dwie klasy pochodne: `Laptop` i `Desktop`. Klasa `Laptop` powinna dodatkowo posiadać pole `weight`, a klasa `Desktop` pole `caseType`. Wszystkie klasy powinny posiadać odpowiednie gettery, settery oraz konstruktory. W klasie testującej, poza pakietem `computers`, stwórz obiekty obu klas pochodnych, nadaj im wartości i wydrukuj je.

9. Stwórz pakiet o nazwie `company`. Wewnątrz tego pakietu utwórz klasę bazową `Employee` z polami: `firstName`, `lastName` i `salary`. Następnie stwórz dwie klasy pochodne: `Manager` i `Developer`. Klasa `Manager` powinna dodatkowo posiadać pole `bonus`, a klasa `Developer` pole `programmingLanguage`. Dodaj odpowiednie metody pozwalające na obliczanie rocznego zarobku (pensja + ewentualne premie). W klasie testującej, poza pakietem `company`, stwórz obiekty obu klas pochodnych, nadaj im wartości i oblicz ich roczne zarobki.

17 Złożone pola w klasie

1. Utwórz klasę `MusicAlbum` z polami `title`, `artist` oraz `ratings` (jako tablica z elementami typu double). Dodaj metodę pozwalającą na dodawanie i usuwanie ocen. Utwórz klasę `RockAlbum`, która dziedziczy po klasie `MusicAlbum`. Klasa `RockAlbum` powinna mieć dodatkowe pole `rockGenre`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

2. Utwórz klasę `Book` z polami `title`, `author` oraz `reviews` (jako tablica z elementami typu double). Dodaj metody pozwalające na dodawanie i usuwanie recenzji. Utwórz klasę `FantasyBook`, która dziedziczy po klasie `Book`. Klasa `FantasyBook` powinna mieć dodatkowe pole `fantasySubgenre`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

3. Utwórz klasę `ComputerGame` z polami `title`, `producer` oraz `ratings` (jako tablica z elementami typu double). Dodaj metody pozwalające na dodawanie i usuwanie ocen. Utwórz klasę `RPGGame`, która dziedziczy po klasie `ComputerGame`. Klasa `RPGGame` powinna mieć dodatkowe pole `gameWorld`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

4. Utwórz klasę `University` z polami `name`, `location` oraz `studyPrograms` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie kierunków studiów. Utwórz klasę `TechnicalUniversity`, która dziedziczy po klasie `University`. Klasa `TechnicalUniversity` powinna mieć dodatkowe pole `numberOfLaboratories`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

5. Utwórz klasę `ArtGallery` z polami `name`, `city` oraz `paintings` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie obrazów. Utwórz klasę `ContemporaryGallery`, która dziedziczy po klasie `ArtGallery`. Klasa `ContemporaryGallery` powinna mieć dodatkowe pole `numberOfInstallations`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

6. Utwórz klasę `Car` z polami `brand`, `model` oraz `engineVariants` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie wariantów silników.

Utwórz klasę `ElectricCar`, która dziedziczy po klasie `Car`. Klasa `ElectricCar` powinna mieć dodatkowe pole `range`. Dodaj konstruktory, metody gettery i settery, metodę `toString()`, `equals()` oraz `hashCode()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

7. Wykonaj poniższe czynności:

- Stwórz klasę `Engine` (Silnik) z polami: `power` (moc), `type` (typ silnika), `serialNumber` (numer seryjny). Dodaj konstruktor parametryczny, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.
- Stwórz klasę `Car` (Samochód). Klasa `Car` powinna mieć pola: `make` (marka), `model` (model), `engine` typu `Engine` (silnik). Dodaj konstruktor parametryczny, który przyjmuje obiekt `Engine` jako parametr, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.

8. Wykonaj poniższe czynności:

- Stwórz klasę `Processor` (Procesor) z polami: `frequency` (częstotliwość), `cores` (liczba rdzeni), `manufacturer` (producent). Dodaj konstruktor parametryczny, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.
- Stwórz klasę `Computer` (Komputer). Klasa `Computer` powinna mieć pola: `brand` (marka), `model` (model), `processor` typu `Processor` (procesor). Dodaj konstruktor parametryczny, który przyjmuje obiekt `Processor` jako parametr, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.

9. Wykonaj poniższe czynności:

- Stwórz klasę `Address` (Adres) z polami: `street` (ulica), `city` (miasto), `zipCode` (kod pocztowy), `country` (kraj). Dodaj konstruktor parametryczny, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.
- Stwórz klasę `UserAccount` (Konto Użytkownika). Klasa `UserAccount` powinna mieć pola: `username` (nazwa użytkownika), `email` (email), `address` typu `Address` (adres). Dodaj konstruktor parametryczny, który przyjmuje obiekt `Address` jako parametr, gettery, settery, oraz metody `toString`, `equals` i `hashCode`.

18 Klasy abstrakcyjne

1. Zdefiniuj abstrakcyjną klasę `WorkTool` z polami `name` typu `String` oraz `productionYear` typu `int`. Dodaj metodę abstrakcyjną `use()`, która będzie symulować użycie narzędzia. Następnie zdefiniuj klasy `Hammer`, `Screwdriver` i `Saw`, które dziedziczą po klasie `WorkTool` i implementują metodę `use()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.
2. Zdefiniuj abstrakcyjną klasę `ComputerGraphic` z polami `width`, `height` typu `int` oraz `fileName` typu `String`. Dodaj abstrakcyjne metody `loadFile()` i `saveFile()`. Następnie zdefiniuj klasy `Bitmap` i `Vector`, które dziedziczą po klasie `ComputerGraphic` i implementują metody `loadFile()` oraz `saveFile()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.
3. Zdefiniuj abstrakcyjną klasę `ElectronicDevice` z polami `manufacturer` typu `String`, `model` typu `String` oraz `productionYear` typu `int`. Dodaj abstrakcyjne metody `turnOn()` i `turnOff()`. Następnie zdefiniuj klasy `Smartphone`, `Television` i `Laptop`, które dziedziczą po klasie `ElectronicDevice` i implementują metody `turnOn()` oraz `turnOff()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.
4. W jednym projekcie wykonaj czynności:

A. Stwórz abstrakcyjną klasę `Product` zawierającą publiczną abstrakcyjną metodę `getPrice()`, która nie przyjmuje argumentów i zwraca `double`. Klasę umieść w pakiecie `store`.

B. Utwórz dwie klasy pochodne od `Product`: `Book` i `Clothing`. W obu klasach nadpisz metodę `getPrice()`. Dla `Book` niech zwraca cenę 29.99, a dla `Clothing` niech zwraca cenę 59.99.

C. W klasie `TestProduct` w pakiecie `store` utwórz tablicę typu `Product` i zainicjuj ją 5 instancjami `Book` i `Clothing`. Iteruj po tablicy wywołując metodę `getPrice()` dla każdego produktu (wyświetl ceny na standardowym wyjściu).

5. W jednym projekcie wykonaj czynności:

A. Stwórz abstrakcyjną klasę `Game` zawierającą publiczną abstrakcyjną metodę `getRating()`, która nie przyjmuje argumentów i zwraca `double`. Klasę umieść w pakiecie `entertainment`.

B. Utwórz dwie klasy pochodne od `Game`: `StrategyGame` i `AdventureGame`. W obu klasach nadpisz metodę `getRating()`. Dla `StrategyGame` niech zwraca ocenę 8.5, a dla `AdventureGame` niech zwraca ocenę 7.3.

C. W klasie `TestGame` w pakiecie `entertainment` utwórz listę tabllicową typu `Game` i zainicjalizuj ją 5 instancjami `StrategyGame` i `AdventureGame`. Iteruj po liście tablicowej wywołując metodę `getRating()` dla każdej gry (wyświetl oceny na standardowym wyjściu).

19 Pola, metody, klasy finalne

1. Wykonaj kolejno czynności:

   A. Stwórz klasę `Planet` z jednym polem finalnym `name`. Spróbuj zmienić wartość pola `name` po jego inicjalizacji. Przeanalizuj wynik.

   B. Dodaj do klasy `Planet` metodę `changeName`, która próbuje zmienić nazwę planety. Jaki jest wynik próby zmiany finalnego pola za pomocą metody?

2. Stwórz klasę `Car` z dwoma polami: `brand` (normalne pole) i `VIN` (pole finalne - Numer Identyfikacyjny Pojazdu). Stwórz kilka obiektów `Car` i spróbuj zmienić pole `VIN` dla każdego z nich. Przeanalizuj wynik.

3. Stwórz klasę `Athlete` z jednym polem finalnym `name` oraz polem `points`. Stwórz tablicę zawodników i spróbuj zmienić pole `name` dla jednego z zawodników. Przeanalizuj wynik.

4. Stwórz klasę `Person` z jednym polem finalnym `PESEL`. Następnie stwórz klasę `Student`, która dziedziczy po klasie `Person`. Czy możesz zmienić pole `PESEL` w klasie `Student`? Co się stanie, jeśli spróbujesz to zrobić? Przeanalizuj wyniki.

5. Stwórz klasę bazową `Electronics` z metodą finalną `turnOn`, która wypisuje "Urządzenie włączone". Następnie stwórz klasę potomną `Television` i spróbuj przesłonić metodę `turnOn`. Zaobserwuj, co się dzieje.

6. Stwórz klasę `Computer` z metodą finalną `boot` oraz zwykłą metodą `launchApplication`. W klasie potomnej `Laptop`, spróbuj przesłonić obie metody. Sprawdź, która z metod pozwoli się przesłonić, a której nie.

7. Utwórz finalną klasę `ImmutableData` z pewnymi atrybutami i metodami. Następnie spróbuj stworzyć klasę potomną `VariableData`, która dziedziczy po klasie `ImmutableData`. Zastanów się, dlaczego nie można dziedziczyć po klasie oznaczonej jako `final`.

8. Stwórz klasę `Game`, która w swoim konstruktorze ma metodę finalną `initialize` (inicializującą pewne dane). Utwórz klasę potomną `RPG`, która próbuje dostosować konstruktor klasy bazowej. Upewnij się, że metoda `initialize` działa poprawnie, mimo że jest oznaczona jako `final`.

9. Utwórz klasę `MathConstant` z finalnym polem `PI`, które jest inicjowane wartością 3.14159. W klasie potomnej `PhysicalConstant`, spróbuj stworzyć metodę, która próbuje zmienić wartość `PI`. Zastanów się, dlaczego wartość finalnego pola nie może zostać zmieniona po jego inicjalizacji.

10. Wykonaj poniższe czynności:

- Stwórz klasę `ImmutableDate` z prywatnymi finalnymi polami: `year` (rok), `month` (miesiąc), `day` (dzień).
- Dodaj konstruktor parametryczny do inicjalizacji wszystkich pól.
- Dodaj publiczne metody `getYear`, `getMonth`, `getDay` do pobierania wartości pól, ale nie dodawaj żadnych metod umożliwiających ich modyfikację.
- Zaimplementuj metody `toString`, `equals` i `hashCode`.
- Sprawdź, czy taka klasa może być uznana jako niemodyfikowalna (inmutable).

11. Wykonaj poniższe czynności:

- Stwórz klasę `ImmutablePoint` z prywatnymi finalnymi polami: `x`, `y`, `z` (współrzędne punktu).
- Dodaj konstruktor parametryczny do inicjalizacji wszystkich pól.
- Dodaj publiczne metody `getX`, `getY`, `getZ` do pobierania wartości pól, ale nie dodawaj żadnych metod umożliwiających ich modyfikację.
- Zaimplementuj metody `toString`, `equals` i `hashCode`.
- Sprawdź, czy taka klasa może być uznana jako niemodyfikowalna (inmutable).

12. Wykonaj poniższe czynności:

- Stwórz klasę `ImmutableBook` z prywatnymi finalnymi polami: `title` (tytuł), `author` (autor), `isbn` (numer ISBN).
- Dodaj konstruktor parametryczny do inicjalizacji wszystkich pól.
- Dodaj publiczne metody `getTitle`, `getAuthor`, `getIsbn` do pobierania wartości pól, ale nie dodawaj żadnych metod umożliwiających ich modyfikację.
- Zaimplementuj metody `toString`, `equals` i `hashCode`.
- Sprawdź, czy taka klasa może być uznana jako niemodyfikowalna (inmutable).

20 Interfejs `Comparable`

1. Napisz klasę `Student`, która zawiera pola: `name` (typu `String`), `averageGrade` (typu `double`) i `yearOfBirth` (typu `int`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Student` były sortowane malejąco według średniej ocen. Stwórz listę tablicową 5 obiektów klasy `Student` i posortuj ją według sprecyzowanego kryterium.

2. Napisz klasę `Employee`, która zawiera pola: `name` (typu `String`), `salary` (typu `double`) i `employmentDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Employee` były sortowane rosnąco według pensji. Stwórz listę tablicową 5 obiektów klasy `Employee` i posortuj ją według sprecyzowanego kryterium.

3. Napisz klasę `Client`, która zawiera pola: `name` (typu `String`), `clientNumber` (typu `int`) i `lastLogin` (typu `Date`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Client` były sortowane malejąco według daty ostatniego logowania. Stwórz listę tablicową 5 obiektów klasy `Client` i posortuj ją według sprecyzowanego kryterium.

4. Napisz klasę `Product`, która zawiera pola: `name` (typu `String`), `price` (typu `double`) i `productionDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Product` były sortowane rosnąco według daty produkcji. Stwórz listę tablicową 5 obiektów klasy `Product` i posortuj ją według sprecyzowanego kryterium.

5. Napisz klasę `Person`, która zawiera pola: `name` (typu `String`), `height` (typu `int`) i `dateOfBirth` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Person` były sortowane malejąco według wzrostu. Stwórz listę tablicową 5 obiektów klasy `Person` i posortuj ją według sprecyzowanego kryterium.

6. Napisz klasę `Book`, która zawiera pola: `title` (typu `String`), `numberOfPages` (typu `int`) i `publicationDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Book` były sortowane malejąco według liczby stron. Stwórz tablicę 4 obiektów klasy `Book` i posortuj ją według sprecyzowanego kryterium.

7. Napisz klasę `Car`, która zawiera pola: `brand` (typu `String`), `mileage` (typu `int`) i `yearOfProduction` (typu `int`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Car` były sortowane rosnąco według przebiegu. Stwórz tablicę 4 obiektów klasy `Car` i posortuj ją według sprecyzowanego kryterium.

8. Napisz klasę `FoodProduct`, która zawiera pola: `name` (typu `String`), `price` (typu `double`) i `expirationDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób,

aby obiekty klasy `FoodProduct` były sortowane rosnąco według daty ważności. Stwórz tablicę 4 obiektów klasy `FoodProduct` i posortuj ją według sprecyzowanego kryterium.

9. Napisz klasę `Music`, która zawiera pola: `title` (typu `String`), `artist` (typu `String`) i `releaseYear` (typu `int`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Music` były sortowane malejąco według roku wydania. Stwórz tablicę 4 obiektów klasy `Music` i posortuj ją według sprecyzowanego kryterium.

10. Napisz klasę `Item`, która zawiera pola: `name` (typu `String`), `weight` (typu `double`) i `price` (typu `double`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Item` były sortowane rosnąco według wagi. Stwórz tablicę 4 obiektów klasy `Item` i posortuj ją według sprecyzowanego kryterium.

11. Napisz klasę `Student`, która zawiera pola: `name` (typu `String`), `averageGrade` (typu `double`) i `yearOfStudy` (typu `int`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Student` były sortowane według jednego kryterium: malejąco według średniej ocen, a przy równości sortowane były rosnąco według roku studiów. Stwórz tablicę 4 obiektów klasy `Student` i posortuj ją według sprecyzowanego kryterium.

12. Napisz klasę `Order`, która zawiera pola: `productName` (typu `String`), `quantity` (typu `int`) i `unitPrice` (typu `double`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Order` były sortowane według jednego kryterium: malejąco według ceny jednostkowej, a przy równości sortowane były rosnąco według ilości. Stwórz listę tablicową 4 obiektów klasy `Order` i posortuj ją według sprecyzowanego kryterium.

13. Napisz klasę `Client`, która zawiera pola: `name` (typu `String`), `balance` (typu `double`) i `lastPurchaseDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Client` były sortowane według jednego kryterium: malejąco według salda, a przy równości sortowane były rosnąco według daty ostatnich zakupów. Stwórz listę tablicową 4 obiektów klasy `Client` i posortuj ją według sprecyzowanego kryterium.

14. Napisz klasę `Course`, która zawiera pola: `name` (typu `String`), `numberOfHours` (typu `int`) i `price` (typu `double`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Course` były sortowane według jednego kryterium: rosnąco według liczby godzin, a przy równości sortowane były malejąco według ceny. Stwórz tablicę 4 obiektów klasy `Course` i posortuj ją według sprecyzowanego kryterium.

15. Napisz klasę `Product`, która zawiera pola: `name` (typu `String`), `price` (typu `double`) i `expirationDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Product` były sortowane według jednego kryterium: malejąco według daty ważności, a przy równości sortowane były rosnąco według ceny. Stwórz listę obiektów klasy `Product` i posortuj ją według sprecyzowanego kryterium. Następnie wyświetl posortowaną listę na ekranie.

16. Napisz klasę `Car`, która zawiera pola: `brand` (typu `String`), `model` (typu `String`) i `registrationNumber` (typu `String`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Car` były sortowane według jednego kryterium: rosnąco według długości numeru rejestracyjnego. Stwórz tablicę 4 obiektów klasy `Car` i posortuj ją według sprecyzowanego kryterium.

17. Napisz klasę `Employee`, która zawiera pola: `firstName` (typu `String`), `lastName` (typu `String`) i `position` (typu `String`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Employee` były sortowane według jednego kryterium: rosnąco według długości nazwiska. Stwórz listę tablicową 4 obiektów klasy `Employee` i posortuj ją według sprecyzowanego kryterium.

18. Napisz klasę `Movie`, która zawiera pola: `title` (typu `String`), `director` (typu `String`) i `genre` (typu `String`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Movie` były sortowane według jednego kryterium: rosnąco według długości tytułu. Stwórz listę tablicową 4 obiektów klasy `Movie` i posortuj ją według sprecyzowanego kryterium.

19. Napisz klasę `Book`, która zawiera pola: `title` (typu `String`), `author` (typu `String`) i `publishDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Book` były sortowane według jednego niestandardowego kryterium: rosnąco według roku wydania. Stwórz tablicę 4 obiektów klasy `Book` i posortuj ją według sprecyzowanego kryterium.

20. Napisz klasę `Product`, która zawiera pola: `name` (typu `String`), `price` (typu `double`) i `productionDate` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Product` były sortowane według jednego niestandardowego kryterium: malejąco według roku produkcji. Stwórz listę tablicową 4 obiektów klasy `Product` i posortuj ją według sprecyzowanego kryterium.

21. Zdefiniuj klasę `Client`, która będzie implementować generyczny interfejs `Comparable`. W klasie tej zadeklaruj prywatne pola `lastName` typu `String` oraz `balance` typu `double`. Implementując metodę `compareTo` interfejsu `Comparable`, porównuj klientów na podstawie ich salda, a w przypadku takiego samego salda - na podstawie nazwiska. Następnie zdefiniuj klasę `Company` dziedziczącą po klasie `Client`. Klasa `Company` ma dodatkowo posiadać prywatne pole `numberOfEmployees` typu `int`. Implementując metodę `compareTo` interfejsu `Comparable` w klasie `Company`, skorzystaj z metody `compareTo` zdefiniowanej w klasie `Client` oraz, w razie potrzeby, uwzględnij pole `numberOfEmployees`. Napisz program `TestClient`, w którym utwórz listę tablicową 5 klientów i firm o nazwie `clientList` posługując się klasą `ArrayList`. W składzie listy powinny wystąpić przynajmniej dwóch klientów o takim samym saldzie i różnym nazwisku oraz dwie firmy o takiej samej liczbie pracowników i różnym saldzie. Wyświetl zawartość listy `clientList`, posortuj ją za pomocą instancyjnej metody `sort` z klasy `ArrayList` i ponownie wyświetl zawartość tej listy.

22. Zdefiniuj klasę `Animal`, która będzie implementować generyczny interfejs `Comparable`. W klasie tej zadeklaruj prywatne pola `species` typu `String` oraz `age` typu `int`. Implementując metodę `compareTo` interfejsu `Comparable`, porównuj zwierzęta na podstawie ich wieku, a w przypadku takiego samego wieku - na podstawie gatunku. Następnie zdefiniuj klasę `Dog` dziedziczącą po klasie `Animal`. Klasa `Dog` ma dodatkowo posiadać prywatne pole `breed` typu `String`. Implementując metodę `compareTo` interfejsu `Comparable` w klasie `Dog`, skorzystaj z metody `compareTo` zdefiniowanej w klasie `Animal` oraz, w razie potrzeby, uwzględnij pole `breed`. Napisz program `TestAnimal`, w którym utwórz listę tablicową 5 zwierząt i psów o nazwie `animalList` posługując się klasą `ArrayList`. W składzie listy powinny wystąpić przynajmniej po 3 obiekty różnych typów.

21 Interfejs `Comparator`

1. Napisz klasę `Osoba` z polami `imie` (`String`), `wiek` (`int`) i `wzrost` (`double`). Napisz klasę implementującą interfejs `Comparator`, która porównuje osoby na podstawie wieku. Stwórz tablicę 5 osób i posortuj ją według wieku.

2. Napisz klasę `Produkt` z polami `nazwa` (`String`), `cena` (`double`) i `dataWaznosci` (`LocalDate`). Napisz klasę implementującą interfejs `Comparator`, która porównuje produkty na podstawie daty ważności. Stwórz listę 5 produktów i posortuj ją według daty ważności.

3. Napisz klasę `Samochod` z polami `marka` (`String`), `rokProdukcji` (`int`) i `cena` (`double`). Napisz klasę implementującą interfejs `Comparator`, która porównuje samochody na podstawie roku produkcji. Stwórz tablicę 5 samochodów i posortuj ją według roku produkcji.

4. Napisz klasę `Pracownik` z polami `imie` (`String`), `pensja` (`double`) i `dataZatrudnienia` (`LocalDate`). Napisz klasę implementującą interfejs `Comparator`, która porównuje pracowników na podstawie pensji. Stwórz tablicę 5 pracowników i posortuj ją według pensji.

5. Napisz klasę `Ksiazka` z polami `tytul` (`String`), `cena` (`double`) i `dataWydania` (`Date`). Napisz klasę implementującą interfejs `Comparator`, która porównuje książki na podstawie daty wydania. Stwórz listę 5 książek i posortuj ją według daty wydania.

6. Napisz klasę `Product` z polami `id` (typu `int`), `name` (typu `String`) oraz `price` (typu `double`). Zaimplementuj generyczny interfejs `Comparator` do porównywania obiektów po polu `price` (od najniższej do najwyższej ceny), a w przypadku równości po polu `id`. Stwórz listę 5 obiektów klasy `Product` i posortuj ją zgodnie z opisanym kryterium.

7. Napisz klasę `Person` z polami `firstName` (typu `String`), `lastName` (typu `String`) oraz `birthDate` (typu `LocalDate`). Zaimplementuj generyczny interfejs `Comparator` do porównywania obiektów po polu `lastName` (alfabetycznie od A do Z), a w przypadku równości po polu `firstName`. Stwórz tablicę 5 obiektów klasy `Person` i posortuj ją zgodnie z opisanym kryterium.

8. Napisz klasę `Order` z polami `id` (typu `int`), `customerName` (typu `String`) oraz `orderDate` (typu `LocalDate`). Zaimplementuj generyczny interfejs `Comparator` do porównywania obiektów po polu `orderDate` (od najwcześniejszej do najpóźniejszej daty), a w przypadku równości po polu `id`. Stwórz listę 5 obiektów klasy `Order` i posortuj ją zgodnie z opisanym kryterium.

9. Napisz klasę `Song` z polami `title` (typu `String`), `artist` (typu `String`) oraz `duration` (typu `int`). Zaimplementuj generyczny interfejs `Comparator` do porównywania obiektów po polu `duration` (od najkrótszej do najdłuższej piosenki), a w przypadku równości po polu `title`. Stwórz tablicę 5 obiektów klasy `Song` i posortuj ją zgodnie z opisanym kryterium.

10. Napisz klasę `Student` z polami `id` (typu `int`), `name` (typu `String`) oraz `averageGrade` (typu `double`). Zaimplementuj generyczny interfejs `Comparator` do porównywania obiektów po polu `averageGrade` (od najwyższej do najniższej średniej ocen), a w przypadku równości po polu `name`. Stwórz listę 5 obiektów klasy `Student` i posortuj ją zgodnie z opisanym kryterium.

11. Napisz klasę `Product` z polami `id` (typu `int`), `name` (typu `String`) oraz `price` (typu `double`). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `PriceComparator` do porównywania obiektów po polu `price` (od najniższej do najwyższej ceny) oraz `NameComparator` do porównywania obiektów po polu `name` (alfabetycznie od A do Z). Stwórz listę 5 obiektów klasy `Product` i posortuj ją zgodnie z oboma kryteriami (najpierw po cenie, a następnie przy równości po nazwie).

12. Napisz klasę `Person` z polami `firstName` (typu `String`), `lastName` (typu `String`) oraz `birthDate` (typu `LocalDate`). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `LastNameComparator` do porównywania obiektów po polu `lastName` (alfabetycznie od A do Z) oraz `BirthDateComparator` do porównywania obiektów po polu `birthDate` (od najstarszej do najmłodszej osoby). Stwórz tablicę 5 obiektów klasy `Person` i posortuj ją zgodnie z oboma kryteriami (najpierw po nazwisku, a następnie przy równości po dacie urodzenia).

13. Napisz klasę `Order` z polami `id` (typu `int`), `customerName` (typu `String`) oraz `orderDate` (typu `LocalDate`). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `OrderDateComparator` do porównywania obiektów po polu `orderDate` (od najwcześniejszej do najpóźniejszej daty) oraz `CustomerNameComparator` do porównywania obiektów po polu `customerName` (alfabetycznie od A do Z). Stwórz listę 5 obiektów klasy `Order` i posortuj ją zgodnie z oboma kryteriami (najpierw po dacie zamówienia, a następnie przy równości po nazwie klienta).

14. Napisz klasę `Song` z polami `title` (typu `String`), `artist` (typu `String`) oraz `duration` (typu `int`). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `DurationComparator` do porównywania obiektów po polu `duration` (od najkrótszej do najdłuższej piosenki) oraz `ArtistTitleComparator` do porównywania obiektów po polu `artist` (alfabetycznie od A do Z) i w przypadku równości po polu `title`. Stwórz tablicę 5 obiektów klasy `Song` i posortuj ją zgodnie z oboma kryteriami (najpierw po długości utworu,a przy równości po artyście i tytule).

15. Napisz klasę `Student` z polami `id` (typu `int`), `name` (typu `String`) oraz `averageGrade` (typu `double`). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `AverageGradeComparator` do porównywania obiektów po polu

`averageGrade` (od najwyższej do najniższej średniej ocen) oraz `IdComparator` do porównywania obiektów po polu `id` (od najniższego do najwyższego identyfikatora). Stwórz listę 5 obiektów klasy `Student` i posortuj ją zgodnie z oboma kryteriami (najpierw po średniej ocen, a następnie przy równości po identyfikatorze).

22 Kopiowanie obiektów

1. Napisz klasę `Student` z trzema polami: `name` (String), `age` (int) i `grade` (double). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Student`, sklonuj go, a następnie zmień ocenę (`grade`) oryginalnego studenta. Wyświetl oceny obu studentów, aby zobaczyć, czy są niezależne.

2. Napisz klasę `Teacher` z trzema polami: `name` (String), `subject` (String) i `experience` (int). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Teacher`, sklonuj go, a następnie zmień doświadczenie (`experience`) oryginalnego nauczyciela. Wyświetl doświadczenie obu nauczycieli, aby zobaczyć, czy są niezależne.

3. Napisz klasę `Car` z trzema polami: `make` (String), `model` (String) i `mileage` (double). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Car`, sklonuj go, a następnie zmień przebieg (`mileage`) oryginalnego samochodu. Wyświetl przebieg obu samochodów, aby zobaczyć, czy są niezależne.

4. Napisz klasę `Smartphone` z trzema polami: `brand` (String), `model` (String) i `productionDate` (typu `Date`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Smartphone`, sklonuj go, a następnie zmień datę produkcji oryginalnego smartfona. Wyświetl datę produkcji obu smartfonów, aby zobaczyć, czy są niezależne.

5. Napisz klasę `Laptop` z trzema polami: `brand` (String), `model` (String) i `purchaseDate` (typu `Date`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Laptop`, sklonuj go, a następnie zmień datę zakupu (`purchaseDate`) oryginalnego laptopa. Wyświetl datę zakupu obu laptopów, aby zobaczyć, czy są niezależne.

6. Napisz klasę `VideoGame` z trzema polami: `title` (String), `genre` (String) i `releaseDate` (typu `Date`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `VideoGame`, sklonuj go, a następnie zmień datę wydania (`releaseDate`) oryginalnej gry. Wyświetl datę wydania obu gier, aby zobaczyć, czy są niezależne.

7. Napisz klasę `CreditCard` z trzema polami: `cardNumber` (String), `holderName` (String) i `expiryDate` (typu `LocalDate`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `CreditCard`, sklonuj go, a następnie zmień datę wygaśnięcia (`expiryDate`) oryginalnej karty kredytowej. Wyświetl datę wygaśnięcia obu kart, aby zobaczyć, czy są niezależne.

8. Napisz klasę `BankAccount` z trzema polami: `accountNumber` (String), `accountHolder` (String) i `openingDate` (typu `LocalDate`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `BankAccount`, sklonuj go, a następnie zmień datę otwarcia (`openingDate`) oryginalnego konta bankowego. Wyświetl datę otwarcia obu kont, aby zobaczyć, czy są niezależne.

9. Napisz klasę `DrivingLicense` z trzema polami: `licenseNumber` (String), `holderName` (String) i `issueDate` (typu `LocalDate`). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `DrivingLicense`, sklonuj go, a następnie zmień datę wydania (`issueDate`) oryginalnego prawa jazdy. Wyświetl datę wydania obu praw jazdy, aby zobaczyć, czy są niezależne.

10. Napisz klasę `Employee` z dwoma polami: `name` (String) i `salaries` (tablica 12 zmiennych typu double, reprezentująca zarobki za każdy miesiąc). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Employee`, sklonuj go, a następnie zmień zarobki na pozycji 5 (czerwiec) oryginalnego pracownika. Wyświetl zarobki obu pracowników, aby zobaczyć, czy są niezależne.

11. Napisz klasę `Athlete` z dwoma polami: `name` (String) i `times` (tablica 5 zmiennych typu double, reprezentująca czas w sekundach potrzebny na przebiegnięcie 100 metrów podczas różnych prób). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Athlete`, sklonuj go, a następnie zmień czas na pozycji 3 oryginalnego sportowca. Wyświetl czasy obu sportowców, aby zobaczyć, czy są niezależne.

12. Napisz klasę `Teacher` z dwoma polami: `name` (String) i `studentsGrades` (tablica 10 zmiennych typu double, reprezentująca oceny każdego z 10 uczniów). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Teacher`, sklonuj go, a następnie zmień ocenę na pozycji 10 oryginalnego nauczyciela. Wyświetl oceny obu nauczycieli, aby zobaczyć, czy są niezależne.

13. Napisz klasę `Employee` z dwoma polami: `name` (String) i `monthlyHours` (lista tablicowa zmiennych typu int, reprezentująca liczbę przepracowanych godzin w każdym miesiącu). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Employee`, sklonuj go, a następnie zmień liczbę godzin na pozycji 5 (czerwiec) oryginalnego pracownika. Wyświetl liczbę godzin obu pracowników, aby zobaczyć, czy są niezależne.

14. Napisz klasę `Athlete` z dwoma polami: `name` (String) i `lapTimes` (lista tablicowa zmiennych typu int, reprezentująca czas w sekundach potrzebny na przebiegnięcie okrążenia podczas różnych prób). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Athlete`, sklonuj go, a następnie zmień czas na pozycji 3 oryginalnego sportowca. Wyświetl czasy obu sportowców, aby zobaczyć, czy są niezależne.

15. Napisz klasę `Teacher` z dwoma polami: `name` (String) i `studentsGrades` (lista tablicowa zmiennych typu int, reprezentująca oceny każdego z uczniów). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()`, aby móc klonować obiekty tej klasy. W metodzie `main()` utwórz obiekt `Teacher`, sklonuj go, a następnie zmień ocenę na pozycji 10 oryginalnego nauczyciela. Wyświetl oceny obu nauczycieli, aby zobaczyć, czy są niezależne.

16. Napisz klasę `Teacher` z polami `name` (String), `age` (int) i `salary` (double). Następnie napisz klasę `HeadTeacher`, która dziedziczy po klasie `Teacher` i dodaje pole `bonus` (double). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()` w obu klasach. W metodzie `main()` pokaż przykład prezentujący poprawność klonowania obiektów tych klas.

17. Napisz klasę `Developer` z polami `name` (String), `age` (int) i `salary` (double). Następnie napisz klasę `SeniorDeveloper`, która dziedziczy po klasie `Developer` i dodaje pole `bonus` (double). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()` w obu klasach. W metodzie `main()` pokaż przykład prezentujący poprawność klonowania obiektów tych klas.

18. Napisz klasę `Nurse` z polami `name` (String), `age` (int) i `salary` (double). Następnie napisz klasę `HeadNurse`, która dziedziczy po klasie `Nurse` i dodaje pole `bonus` (double). Zaimplementuj interfejs `Cloneable` i nadpisz metodę `clone()` w obu klasach. W metodzie `main()` pokaż przykład prezentujący poprawność klonowania obiektów tych klas.

19. Wykonaj poniższe czynności:

- Stwórz klasę `Engine` (Silnik) z polami: `power` (moc), `type` (typ silnika), `serialNumber` (numer seryjny). Dodaj konstruktor parametryczny oraz zaimplementuj interfejs `Cloneable`.
- Stwórz klasę `Car` (Samochód). Klasa `Car` powinna mieć pola: `make` (marka), `model` (model), `engine` typu `Engine` (silnik). Dodaj konstruktor parametryczny oraz zaimplementuj interfejs `Cloneable`.

23 Interfejsy

1. Napisz interfejs o nazwie `LoudAnimal`, który będzie miał jedną metodę o nazwie `makeNoise()`. Następnie stwórz dwie klasy: `Dog` i `Cat`, które będą implementować ten interfejs. Dla każdej klasy zaimplementuj metodę `makeNoise()`, tak aby wydrukowała ona odpowiedni dźwięk zwierzęcia.

2. Stwórz interfejs o nazwie `MyComparator`, który będzie zawierał metodę `compare(int a, int b): int`. Metoda ta powinna zwrócić -1, jeśli `a < b`, 0 jeśli `a == b` i 1 jeśli `a > b`. Następnie stwórz klasę `TestMyComparator`, która będzie implementować ten interfejs. W klasie tej zaimplementuj metodę `compare` zgodnie z jej opisem.

3. Załóżmy, że mamy interfejs `MusicPlayer` z metodami `turnOn()`, `turnOff()` i `nextTrack()`. Stwórz klasę `Radio`, która będzie implementować ten interfejs. W metodzie `turnOn()` powinien zostać wydrukowany komunikat "Radio włączone", w metodzie `turnOff()` - "Radio wyłączone", a w `nextTrack()` - "Zmieniono stację radiową".

4. Napisz interfejs `Converter` z trzema abstrakcyjnymi metodami: `convertToEuro(double amount)` zwracającą `double`, `convertToUSD(double amount)` zwracającą `double` oraz `getConversionRate(String currency)` przyjmującą nazwę waluty jako `String` i zwracającą `double`. Stwórz dwie klasy: `CurrencyConverter` i `UnitConverter`, które implementują ten interfejs. W osobnej klasie `ConverterTest`, przetestuj implementację metod dla obiektów obu klas.

5. Utwórz interfejs `Authentication` z trzema metodami abstrakcyjnymi: `login(String username, String password)` zwracającą `boolean`, `logout()` zwracającą `void` oraz `resetPassword(String username, String oldPassword, String newPassword)` zwracającą `boolean`. Stwórz dwie klasy `UserAuthentication` i `AdminAuthentication`, które implementują ten interfejs. W klasie `AuthenticationTest` przetestuj implementacje metod dla obiektów obu klas.

6. Zaprojektuj interfejs `Sensor` z trzema metodami abstrakcyjnymi: `readValue()` zwracającą `double`, `getStatus()` zwracającą `String` oraz `reset()` zwracającą `void`. Stwórz dwie klasy `TemperatureSensor` i `PressureSensor`, które implementują ten interfejs. W klasie `SensorTest` przetestuj działanie metod dla obiektów z obu klas.

7. Opracuj interfejs `MediaPlayer` z trzema abstrakcyjnymi metodami: `play(String trackName)` zwracającą `void`, `pause()` zwracającą `void` oraz `getCurrentTrack()` zwracającą `String`. Stwórz dwie klasy `AudioPlayer` i `VideoPlayer`, które implementują

ten interfejs. W osobnej klasie `MediaPlayerTest` sprawdź działanie metod dla obiektów z obu klas.

8. Stwórz interfejs `Storage` z trzema metodami abstrakcyjnymi: `save(String data)` zwracającą `boolean`, `delete(String identifier)` zwracającą `boolean` oraz `retrieve(String identifier)` zwracającą `String`. Zaprojektuj dwie klasy `FileStorage` i `DatabaseStorage`, które implementują ten interfejs. W klasie `StorageTest` wykonaj testy metod dla obiektów z obu klas.
9. Wykonaj poniższe czynności:

- Napisz interfejs `DataProcessor` z dwoma metodami abstrakcyjnymi: `processData(String data)` zwracającą `String` i `isValid(String data)` zwracającą `boolean`.
- Stwórz klasę `TextProcessor`, która implementuje `DataProcessor`. W metodzie `processData` zwróć dane w odwróconej kolejności, a w metodzie `isValid` sprawdź, czy dane nie są pustym napisem. Zabezpiecz metody przed nullem.
- Stwórz drugą klasę `NumberProcessor`, która również implementuje `DataProcessor`. W metodzie `processData` zwróć dane z dodatkowym tekstem na początku "Processed:", a w metodzie `isValid` sprawdź, czy dane są liczbą.
- W klasie `TestDataProcessor` stwórz obiekty obu klas i przetestuj ich metody.

Wskazówka: W celu sprawdzenia, że string jest liczbą możesz użyć wyrażenia regularne:

```
str.matches("-?\\d+(\\.\\d+)?")
```

10. Wykonaj poniższe czynności:

- Utwórz interfejs `VehicleManager` z dwoma metodami abstrakcyjnymi: `startEngine()` zwracającą `String` i `getFuelLevel()` zwracającą `int`.
- Stwórz klasę `Car`, implementującą `VehicleManager`. W metodzie `startEngine` zwróć "Silnik samochodu uruchomiony", a w `getFuelLevel` zwróć wartość 50.
- Stwórz klasę `Motorcycle`, również implementującą `VehicleManager`. W `startEngine` zwróć "Silnik motocykla uruchomiony", a w `getFuelLevel` zwróć wartość 30.
- W klasie `VehicleManagerTest` stwórz obiekty obu klas i przetestuj ich metody.

11. Wykonaj poniższe czynności:

- Stwórz interfejs `AnimalSound` z jedną metodą `makeSound()`.
- Rozszerz ten interfejs, tworząc interfejs `DomesticAnimalSound`, który dodaje metodę `makeHappySound()`.
- Stwórz klasę `Dog`, która implementuje `DomesticAnimalSound`. Metoda `makeSound()` powinna zwracać string "Woof", a `makeHappySound()` - "Wag tail".
- W osobnej klasie testującej (`TestAnimals`), utwórz obiekt `Dog` i wywołaj obie metody.

12. Wykonaj poniższe czynności:

- Stwórz interfejs `PowerControl` z metodą `turnOn()`.
- Rozszerz ten interfejs, tworząc `AdvancedPowerControl`, który dodaje metodę `setPowerSavingMode()`.
- Stwórz klasę `SmartLamp`, implementującą `AdvancedPowerControl`. Metoda `turnOn()` powinna aktywować lampę, a `setPowerSavingMode()` - przełączać ją w tryb oszczędzania energii (metody mają wyświetlać odpowiednie komunikaty).
- W osobnej klasie testującej (`TestDevices`), utwórz obiekt `SmartLamp` i wywołaj obie metody.

13. Wykonaj poniższe czynności:

- Stwórz interfejs `Drawable` z metodą `draw()`.
- Rozszerz ten interfejs, tworząc `ColorDrawable`, który dodaje metodę `setColor(String color)`.
- Stwórz klasę `Circle`, implementującą `ColorDrawable`. Metoda `draw()` powinna rysować koło, a `setColor()` - zmieniać kolor koła (metody mają wyświetlać odpowiednie komunikaty).
- W osobnej klasie testującej (`TestDrawing`), utwórz obiekt `Circle`, ustaw kolor i narysuj koło za pomocą obu metod.

14. Stwórz interfejs `Drawable` z:

- Metodą abstrakcyjną `draw()`.
- Metodą domyślną `display()` wyświetlającą informację "Displaying Drawable".
- Metodą statyczną `getType()` zwracającą `String` "Drawable Type".

Stwórz klasy `Circle` i `Rectangle`, które implementują `Drawable`. `draw()` w `Circle` powinno wyświetlać "Drawing Circle", a w `Rectangle` - "Drawing Rectangle". Stwórz klasę testującą `DrawableTester`. Utwórz obiekty `Circle` i `Rectangle`, wywołaj dla nich `draw()` i `display()`, oraz statycznie `Drawable.getType()`.

15. Stwórz interfejs `SoundPlayer` z:

- Metodą abstrakcyjną `playSound()`.
- Metodą domyślną `stopSound()` wyświetlającą informację "Sound Stopped".
- Metodą statyczną `getDeviceType()` zwracającą `String` "Sound Device".

Stwórz klasy `MusicPlayer` i `Radio`, które implementują `SoundPlayer`. `playSound()` w `MusicPlayer` powinno wyświetlać "Playing Music", a w `Radio` - "Playing Radio". Stwórz klasę testującą `SoundTester`. Utwórz obiekty `MusicPlayer` i `Radio`, wywołaj dla nich `playSound()` i `stopSound()`, oraz statycznie `SoundPlayer.getDeviceType()`.

16. Stwórz interfejs `MemoryManager` z:

- Metodą abstrakcyjną `allocateMemory(int size)`.

- Metodą domyślną `freeMemory()` wyświetlającą informację "Memory Freed".
- Metodą statyczną `getMemoryType()` zwracającą `String` "Memory Type".

Stwórz klasy `RAMManager` i `DiskManager`, które implementują `MemoryManager`. `allocateMemory(int size)` w `RAMManager` powinno wyświetlać "Allocating RAM Memory", a w `DiskManager` - "Allocating Disk Space". Stwórz klasę testującą `MemoryTester`. Utwórz obiekty `RAMManager` i `DiskManager`, wywołaj dla nich `allocateMemory(int size)` i `freeMemory()`, oraz statycznie `MemoryManager.getMemoryType()`.

17. Stwórz interfejs `VATCalculator` z:

- Polem `double vatRate` ustawionym na wartość standardowej stawki VAT (np. 23%).
- Metodą statyczną `calculateWithVAT(double price)`, która oblicza i zwraca cenę produktu z doliczonym VAT.

Stwórz klasę testującą `VATTest`, która wywołuje metodę `calculateWithVAT(double price)` z przykładową ceną produktu i wyświetla wynik.

24 Wyjątki

1. Napisz program, który definiuje metodę `checkAge(int age)`. Metoda ta powinna rzucić wyjątek `IllegalArgumentException` z odpowiednim komunikatem, jeśli podany wiek jest mniejszy niż 18. W głównej metodzie programu (`main`) wywołaj `checkAge` z różnymi wartościami i obsłuż wyjątek, wyświetlając stosowny komunikat dla użytkownika.

2. Napisz program, który prosi użytkownika o wpisanie dwóch liczb, a następnie dzieli pierwszą liczbę przez drugą. Program powinien obsługiwać dwa rodzaje wyjątków: `ArithmeticException` w przypadku dzielenia przez zero i `InputMismatchException`, gdy użytkownik wprowadzi coś innego niż liczby. W obu przypadkach należy wyświetlić stosowny komunikat błędu i poprosić użytkownika o ponowne wprowadzenie danych. Wykorzystaj typ `int`.

3. Zaprojektuj i zaimplementuj klasę wyjątku `NiepoprawnyFormatDanychException`, która będzie rozszerzać klasę `Exception`. Następnie napisz metodę `sprawdzFormatDanych(String dane)`, która rzuci wyjątek `NiepoprawnyFormatDanychException`, jeśli podany ciąg znaków nie odpowiada określonemu wzorcowi (np. nie jest adresem e-mail). W metodzie `main` przetestuj działanie tej metody, obsługując wyjątek i informując użytkownika o błędzie.

25 Programowanie generyczne

1. Stwórz prostą klasę generyczną `Box`, która może przechowywać obiekt dowolnego typu. Klasa powinna zawierać metodę `set`, aby ustawić obiekt, oraz metodę `get`, aby go pobrać.
2. Napisz generyczną metodę `isEqual`, która przyjmuje dwa dowolne obiekty tego samego typu i zwraca `true`, jeśli są one równe, w przeciwnym razie `false`.
3. Stwórz klasę generyczną `Counter<T>`, która będzie zliczać ilość dodanych elementów określonego typu. Klasa powinna mieć metodę `add(T element)`, która dodaje element do wewnętrznej struktury, oraz metodę `getCount()`, która zwraca liczbę dodanych elementów.
4. Stwórz klasę generyczną `Triple<T, U, V>`, która może przechowywać trzy obiekty różnych typów. Zaimplementuj metody `getFirst()`, `getSecond()` i `getThird()` do pobierania odpowiednio pierwszego, drugiego i trzeciego elementu.
5. Napisz generyczną metodę `max`, która przyjmuje tablicę elementów typu porównywalnego (implementujących interfejs `Comparable<T>`) i zwraca element o najwyższej wartości. Uwzględnij obsługę przypadku pustej tablicy.
6. Zdefiniuj generyczny interfejs `Stack<T>` z metodami `push(T item)`, `T pop()`, `T peek()` i `boolean isEmpty()`. Stwórz klasę implementującą ten interfejs, która będzie reprezentować stos przechowujący elementy dowolnego typu.
7. Stwórz klasę generyczną `Storage<T>`, która przechowuje pojedynczy obiekt dowolnego typu. Klasa powinna mieć metody `store(T item)`, która zapisuje obiekt, oraz `T retrieve()`, która zwraca przechowywany obiekt.
8. Napisz generyczną metodę `printArray`, która przyjmuje tablicę elementów dowolnego typu i wypisuje wszystkie elementy tej tablicy na standardowe wyjście. Metoda powinna być w stanie obsłużyć tablice każdego typu obiektów.
9. Stwórz klasę generyczną `Pair`, która przechowuje dwie wartości dowolnego typu. Klasa powinna mieć dwie metody: `getFirst()`, która zwraca pierwszy element pary, i `getSecond()`, która zwraca drugi element pary.
10. Napisz generyczną metodę `swap`, która przyjmuje tablicę dowolnego typu oraz dwie liczby całkowite reprezentujące indeksy. Metoda powinna zamienić miejscami elementy tablicy na podanych indeksach.

11. Stwórz generyczną klasę `GenericQueue<T>`, która implementuje prostą kolejkę. Klasa powinna mieć metody `enqueue(T element)`, która dodaje element do kolejki, i `dequeue()`, która usuwa i zwraca element z początku kolejki.

12. Napisz statyczną metodę generyczną `swap`, która przyjmuje tablicę dowolnego typu i dwa indeksy, a następnie zamienia miejscami elementy w tej tablicy pod wskazanymi indeksami. Metoda powinna działać dla tablicy każdego typu. Przykładowe wywołanie metody: `swap(myArray, 0, 2);`, gdzie `myArray` to tablica typu `Integer[]` lub dowolnego innego typu. Zabezpiecz metodę tak, aby nie można było jej wywołać z indeksami spoza zakresu tablicy.

13. Utwórz statyczną metodę generyczną `reverseArray`, która przyjmuje tablicę elementów typu `T` i odwraca kolejność jej elementów. Metoda powinna modyfikować przekazaną tablicę, nie zwracając nowej. Sprawdź działanie metody na tablicach różnych typów, takich jak `Character`, `Boolean` oraz własnych typów obiektowych. Zabezpiecz metodę tak, aby nie można było jej wywołać z tablicą `null`.

14. Napisz statyczną metodę generyczną `minValue`, która przyjmuje tablicę elementów typu generycznego `T`, gdzie `T` rozszerza `Comparable<T>`. Metoda powinna zwracać najmniejszy element z tablicy. Przetestuj tę metodę na tablicach zawierających różne typy porównywalnych obiektów, takie jak `Integer`, `Double`, czy `String`. Zabezpiecz metodę tak, aby nie można było jej wywołać z tablicą o zerowej liczbie elementów. Dodaj klasę `Person` z polami `name` i `age` i przetestuj metodę `minValue` na tablicy obiektów `Person`.

15. Napisz statyczną metodę generyczną `maxValue`, która przyjmuje tablicę elementów typu generycznego `T`, gdzie `T` rozszerza `Comparable<T>`. Metoda powinna zwracać największy element z tablicy. Upewnij się, że metoda nie akceptuje pustej tablicy (o zerowej liczbie elementów). Przetestuj metodę na tablicach zawierających różne typy porównywalnych obiektów, jak `Integer`, `Float`, czy `String`. Stwórz klasę `Vehicle` z polami `model` i `speed`, implementującą generyczny `Comparable`, i przetestuj metodę `maxValue` na tablicy obiektów `Vehicle`.

16. Utwórz statyczną metodę generyczną `sortAndReturnFirst`, która przyjmuje tablicę elementów typu generycznego `T`, gdzie `T` rozszerza `Comparable<T>`. Metoda powinna sortować tablicę i zwracać jej pierwszy element. Zabezpiecz metodę przed wywołaniem z pustą tablicą (o zerowej liczbie elementów). Przetestuj tę metodę na różnych typach danych, w tym na nowo utworzonej klasie `Book` z polami `title` i `author`. Klasa `Book` powinna implementować generyczny interfejs `Comparable` na podstawie pola `title` zgodnie z porządkiem naturalnym.

17. Napisz statyczną metodę generyczną `printArray`, która przyjmuje jako argument tablicę elementów typu generycznego `T` i drukuje wszystkie elementy tablicy. Przetestuj tę metodę na tablicy typu `Integer` i `Double`. Następnie spróbuj przetestować metodę na tablicy typu prostego, np. `int` lub `double`, i zanotuj wynikające z tego błędy kompilacji. Wyjaśnij, dlaczego Java nie pozwala na użycie typów prostych w kontekście typów generycznych.

18. Utwórz statyczną metodę generyczną `sumElements`, która przyjmuje kolekcję elementów typu generycznego `T` i zwraca ich sumę. Przetestuj tę metodę na kolekcji typu `Integer` oraz `Float`. Następnie spróbuj użyć tej metody z kolekcją typów prostych, np. `int` lub `float`, i zobacz, jakie błędy kompilacji się pojawią. Wykorzystaj to do wyjaśnienia różnicy między typami prostymi a obiektowymi w Javie.

19. Zaprojektuj statyczną metodę generyczną `compareElements`, która przyjmuje dwa argumenty typu generycznego `T` i zwraca wartość `boolean` wskazującą, czy są one równe. Przetestuj metodę na parze obiektów `Integer`, `String` oraz `Character`. Następnie spróbuj przetestować tę samą metodę na typach prostych, takich jak `int` czy `char`, i opisz napotkane problemy związane z próbą użycia typów prostych w generykach.

20. Napisz statyczną metodę generyczną `printObjectInfo`, która przyjmuje argument typu generycznego `T` z górnym ograniczeniem typu `Object`. Metoda powinna drukować klasę obiektu, jego hashcode oraz wynik metody `toString()`. Przetestuj tę metodę na różnych typach obiektów, jak `String`, `Integer`, własnych klasach np. `Car`, `Animal`, gdzie każda klasa ma własne unikalne pola i metody.

21. Utwórz dwie klasy: `Animal` (Zwierzę) i `Dog` (Pies), gdzie `Dog` dziedziczy po `Animal`. Następnie napisz statyczną metodę generyczną `findMax`, która przyjmuje dwa argumenty: `element1` i `element2` typu `extends Animal`. Metoda powinna zwracać element większy według własnie zdefiniowanego kryterium porównania. W implementacji porównaj elementy bazując na wybranym przez siebie atrybucie, na przykład wieku.

22. Zdefiniuj klasy `Car` (Samochód) i `ElectricCar` (Samochód Elektryczny), gdzie `ElectricCar` dziedziczy po `Car`. Napisz statyczną metodę generyczną `compareObjects`, która przyjmuje dwa argumenty: `object1` i `object2` typu `extends Car`. Metoda ma zwracać wartość `true`, jeśli obiekty są tego samego typu, w przeciwnym wypadku `false`. Użyj metody `getClass()` do porównania klas obiektów.

23. Stwórz klasę generyczną `Pair<T>` która przechowuje dwa obiekty tego samego typu. Utwórz dwie klasy: `Animal` (Zwierzę) i `Dog` (Pies), gdzie `Dog` dziedziczy po `Animal`. Następnie napisz statyczną metodę generyczną `findMax`, która przyjmuje `Pair<? extends Animal>`. Metoda powinna zwracać element większy według własnie zdefiniowanego kryterium porównania.

24. Stwórz klasę generyczną `Triple<T>`, która przechowuje trzy obiekty tego samego typu. Następnie utwórz dwie klasy: `Bird` i `Eagle`, gdzie `Eagle` dziedziczy po `Bird`. Potem napisz statyczną metodę generyczną `findMin`, która przyjmuje `Triple<? extends Bird>`. Ta metoda powinna zwracać element mniejszy na podstawie kryterium porównania zdefiniowanego przez ciebie.

25. Stwórz klasę generyczną `Triple<T>`, która przechowuje trzy obiekty tego samego typu. Następnie utwórz dwie klasy: `Bird` i `Eagle`, gdzie `Eagle` dziedziczy po `Bird`. Potem napisz statyczną metodę generyczną `findMin`, która przyjmuje `Triple<? extends Bird>`.

Ta metoda powinna zwracać element mniejszy na podstawie kryterium porównania zdefiniowanego przez ciebie.

26. Stwórz klasę generyczną `ElementPair<T>`, która przechowuje dwa obiekty tego samego typu. Utwórz dwie klasy: `Shape` i `Circle`, gdzie `Circle` dziedziczy po `Shape`. Następnie napisz statyczną metodę generyczną `findLargest`, która przyjmuje `ElementPair<? extends Shape>`. Metoda powinna zwracać element większy według własnie zdefiniowanego kryterium porównania.

27. Stwórz klasę generyczną `DoubleElement<T>`, która przechowuje dwa obiekty tego samego typu. Utwórz dwie klasy: `Person` i `Student`, gdzie `Student` dziedziczy po `Person`. Następnie napisz statyczną metodę generyczną `findYoungest`, która przyjmuje `DoubleElement<? extends Person>`. Metoda powinna zwracać element mniejszy według własnego kryterium porównania, na przykład wieku.

28. Stwórz klasę generyczną `Pair<T>` która przechowuje dwa obiekty tego samego typu. Utwórz dwie klasy: `Animal` (Zwierzę) i `Dog` (Pies), gdzie `Dog` dziedziczy po `Animal`. Klasa `Dog` ma posiadać prywatne pole `age`, które nie posiada klasa `Animal`. Następnie napisz statyczną metodę generyczną `findMinMaxAge`, która przyjmuje jako argument tablicę obiektów typu `Dog` oraz `Pair<?super Dog> result`. Metoda powinna ma zapisać (jako obiekty typu `Dog`) najmniejszy i najmniejszy (pod kątem wieku) psa z tablicy w drugim argumencie metody. Zrób to bezpośrednio bez używania interfejsów `Comparable` czy `Comparator`.

29. Stwórz klasę generyczną `Pair<T>`, która przechowuje dwa obiekty tego samego typu. Następnie utwórz dwie klasy: `Plant` i `Tree`, gdzie `Tree` dziedziczy po `Plant`. Klasa `Tree` powinna posiadać prywatne pole `height`, którego nie posiada klasa `Plant`. Następnie napisz statyczną metodę generyczną `findMinMaxHeight`, która przyjmuje jako argument tablicę obiektów typu `Tree` oraz `Pair<? super Tree> result`. Metoda powinna zapisać (jako obiekty typu Tree) najniższe i najwyższe (pod kątem wysokości) drzewo z tablicy w drugim argumencie metody. Wykorzystaj też generyczny interfejs `Comparable`.

26 Kolekcje

26.1 Interfejs `Collection`

A1. Napisz statyczną metodę `printUnique(Collection<T> items)`, która przyjmuje generyczny interfejs Collection jako argument i wyświetla na ekranie każdy unikalny element z tej kolekcji dokładnie raz.

A2. Stwórz statyczną metodę `countOccurrences(Collection<T> items, T element)`, która zwraca liczbę wystąpień danego elementu w podanej kolekcji. Funkcja powinna działać dla dowolnego typu obiektów przechowywanych w kolekcji.

A3. Zaimplementuj metodę `removeEveryOther(Collection<T> items)`, która usuwa co drugi element z przekazanej kolekcji. Metoda powinna modyfikować oryginalną kolekcję, nie tworząc jej kopii.

26.2 Interfejs `Iterable`

B1. Napisz metodę `reversePrint(Iterable items)`, która przyjmuje generyczny interfejs Iterable jako argument i wyświetla na ekranie elementy tej sekwencji w odwrotnej kolejności, niż zostały one przekazane.

B2. Stwórz funkcję `findMax(Iterable numbers)`, która przeszukuje kolekcję typu Iterable zawierającą liczby i zwraca największą liczbę. Zakładamy, że elementy kolekcji są obiektami klasy `Comparable`.

B3. Zaimplementuj metodę `countElements(Iterable items, Object element)`, która zlicza ile razy dany element pojawił się w kolekcji implementującej interfejs Iterable. Metoda powinna porównywać elementy przy użyciu metody `equals`.

26.3 Lista tablicowa `ArrayList`

C1. Stwórz metodę `mergeLists`, która przyjmuje dwie generyczne `ArrayList<T>` i zwraca nową `ArrayList<T>`, będącą połączeniem elementów z obu list. Upewnij się, że kolejność elementów z oryginalnych list jest zachowana w wynikowej liście.

C2. Napisz funkcję `removeDuplicates`, która przyjmuje `ArrayList<T>` i zwraca nową listę, z której usunięto wszystkie duplikaty, pozostawiając tylko unikalne elementy. Kolejność zachowanych elementów powinna odpowiadać ich pierwszemu wystąpieniu na oryginalnej liście.

C3. Zaimplementuj metodę `countOccurrences`, która przyjmuje `ArrayList<T>` i element typu `T`, a następnie zwraca liczbę wystąpień tego elementu w podanej liście.

26.4 Lista powiązana `LinkedList`

D1. Napisz metodę `isPalindrome`, która przyjmuje generyczną listę powiązaną (`LinkedList<T> list`) i zwraca `true`, jeśli lista jest palindromem, a `false` w przeciwnym przypadku. Lista jest palindromem, gdy czytana od przodu i od tyłu jest taka sama. Metoda powinna być jak najbardziej wydajna.

D2. Napisz metodę `findCommonElements`, która przyjmuje dwie generyczne listy powiązane (`LinkedList<T> list1` i `LinkedList<T> list2`) i zwraca nową listę zawierającą elementy, które występują zarówno w `list1`, jak i `list2`. Elementy w zwróconej liście powinny być unikalne i nie muszą być posortowane. Metoda nie powinna modyfikować wejściowych list.

26.5 Zbiór bazujący na tablicy skrótów `HashSet`

E1. Napisz metodę `findUniqueElements`, która przyjmuje generyczną listę (`List<T> list`) i zwraca `HashSet<T>`, który zawiera tylko unikalne elementy z tej listy. Metoda powinna skutecznie eliminować duplikaty.

E2. Napisz metodę `hasCommonElements`, która przyjmuje dwa generyczne zbiory (`HashSet<T> set1` i `HashSet<T> set2`) i zwraca `true`, jeśli oba zbiory mają przynajmniej jeden wspólny element, oraz `false` w przeciwnym przypadku.

E3. Napisz metodę `unionSets`, która przyjmuje dwie generyczne kolekcje typu `HashSet<T>` (`HashSet<T> set1` i `HashSet<T> set2`) i zwraca nowy zbiór (`HashSet<T>`), który jest zbiorem unii obu wejściowych zbiorów. Wynikowy zbiór powinien zawierać wszystkie elementy, które występują w `set1`, `set2`, lub w obu tych zbiorach.

26.6 `TreeSet`

F1. Napisz metodę `findElementsInRange`, która przyjmuje `TreeSet<T>` oraz dwie wartości graniczne `T lowerBound` i `T upperBound`. Metoda powinna zwracać `TreeSet<T>`, który zawiera wszystkie elementy z pierwotnego zbioru, które mieszczą się w przedziale między `lowerBound` a `upperBound` (włącznie z obiema granicami).

F2. Napisz metodę `removeMinMax`, która przyjmuje `TreeSet<T>` i modyfikuje go przez usunięcie najmniejszego i największego elementu. Metoda powinna zwracać parę (np. klasę `Pair` lub dwuelementową tablicę/listę) zawierającą usunięte wartości. Jeśli zbiór ma mniej niż dwa elementy, metoda powinna zwrócić odpowiednie wartości nullowe lub wskazywać na błąd.

F3. Napisz metodę `findClosestElement`, która przyjmuje `TreeSet<T>` i wartość `T target`. Metoda powinna zwracać element zbioru, który jest najbliżej wartości `target`. W przypadku, gdy dwa elementy są równie blisko, metoda może zwrócić dowolny z nich. Zakładamy, że typ `T` pozwala na porównywanie wartości (np. poprzez implementację interfejsu `Comparable<T>`).

26.7 `Queue`

G1. Napisz metodę `reverseQueue`, która przyjmuje generyczną kolejkę (`Queue<T> queue`) i odwraca kolejność jej elementów. Metoda powinna wykonywać operacje odwracania w miejscu, nie używając dodatkowych kolekcji.

G2. Napisz metodę `simulateSupermarketQueue`, która przyjmuje generyczną kolejkę (`Queue<Customer> customers`) i dodatnią liczbę całkowitą `n`, reprezentujący liczbę dostępnych kas. Klasa `Customer` zawiera czas obsługi klienta. Metoda powinna zwrócić całkowity czas potrzebny do obsłużenia wszystkich klientów w kolejce, przy założeniu, że każda kasa obsługuje jednego klienta na raz i wybiera następnego klienta z przodu kolejki, gdy tylko zostanie zwolniona.

26.8 `Deque`

H1. Napisz metodę `isSymmetric`, która przyjmuje generyczny Deque (`Deque<T> deque`) i zwraca `true`, jeśli elementy w Deque są ułożone symetrycznie (tj. kolejność elementów jest taka sama, gdy czytamy je od przodu do tyłu i od tyłu do przodu). W przeciwnym przypadku metoda zwraca `false`.

H2. Napisz metodę `removeEverySecondElement`, która przyjmuje generyczny Deque (`Deque<T> deque`) i usuwa z niego co drugi element, zaczynając od drugiego elementu w kolejności, w której elementy zostały dodane. Metoda powinna modyfikować oryginalny Deque, nie zwracając nic.

H3. Napisz metodę `swapEnds`, która przyjmuje generyczny Deque (`Deque<T> deque`) i zamienia miejscami pierwszy element z ostatnim. Jeśli Deque zawiera mniej niż dwa elementy, metoda nie powinna nic robić. Metoda powinna modyfikować oryginalny Deque, nie zwracając nic.

26.9 Kolejka priorytetowa `PriorityQueue`

I1. Napisz metodę `mergePriorityQueues`, która przyjmuje dwie generyczne PriorityQueues (`PriorityQueue<T> queue1` i `PriorityQueue<T> queue2`). Metoda powinna zwrócić nową PriorityQueue zawierającą wszystkie elementy z obu wejściowych kolejek. Zakładamy, że elementy w obu kolejnych mogą być porównywane między sobą.

26.10 `Map`

J1. Napisz metodę `reverseMap`, która przyjmuje generyczną mapę (`Map<K, V> map`) i zwraca nową mapę (`Map<V, K>`), gdzie każdy klucz staje się wartością, a każda wartość kluczem. Jeśli oryginalna mapa zawiera powtarzające się wartości, zachowaj tylko ostatnią parę klucz-wartość odwracając mapę.

J2. Napisz metodę `findKeysWithMaxValue`, która przyjmuje generyczną mapę (`Map<K, V> map`), gdzie wartości są porównywalne (implementują `Comparable<V>`). Metoda powinna zwracać listę wszystkich kluczy, które są powiązane z największą wartością w mapie.

J3. Napisz metodę `groupByValue`, która przyjmuje generyczną mapę (`Map<K, V> map`) i zwraca nową mapę (`Map<V, List<K>>`), gdzie każda wartość z oryginalnej mapy staje się kluczem w nowej mapie, a jej wartością jest lista kluczy, które były z nią powiązane w oryginalnej mapie.

26.11 `HashMap`

K1. Napisz metodę `countValueOccurrences`, która przyjmuje generyczną HashMap (`HashMap<K, V> map`) i zwraca nową HashMap (`HashMap<V, Integer>`), gdzie każdy klucz to wartość z oryginalnej mapy, a wartość to liczba wystąpień tej wartości w oryginalnej mapie.

K2. Napisz metodę `swapKeysAndValues`, która przyjmuje generyczną HashMap (`HashMap<K, V> map`) i zwraca nową HashMap (`HashMap<V, K>`), gdzie każdy klucz z oryginalnej mapy staje się wartością, a każda wartość kluczem. Zakładamy, że wszystkie wartości w oryginalnej mapie są unikalne.

K3. Napisz metodę `aggregateValuesByKey`, która przyjmuje generyczną HashMap (`HashMap<K, Integer> map`) oraz klucz (`K key`) i wartość (`Integer value`). Metoda powinna dodać wartość do obecnej wartości skojarzonej z kluczem. Jeśli klucz nie istnieje w mapie, powinien zostać dodany z podaną wartością. Metoda nie zwraca nic, modyfikując oryginalną mapę.

K4. Napisz metodę `compareMaps`, która przyjmuje dwie generyczne HashMap (`HashMap<K, V> map1` i `HashMap<K, V> map2`) i zwraca `true`, jeśli obie mapy mają dokładnie te same pary

klucz-wartość, oraz `false` w przeciwnym przypadku. Metoda powinna porównywać zarówno klucze, jak i wartości.

26.12 `TreeMap`

L1. Napisz metodę `subMapInRange`, która przyjmuje generyczną TreeMap (`TreeMap<K, V> map`), a także dwie wartości kluczy `K startKey` i `K endKey`. Metoda powinna zwrócić nową TreeMap, która zawiera wszystkie pary klucz-wartość z oryginalnej mapy, których klucze mieszczą się w zakresie od `startKey` do `endKey`, włącznie.

L2. Napisz metodę `reverseOrderMap`, która przyjmuje generyczną TreeMap (`TreeMap<K, V> map`). Metoda powinna zwrócić nową TreeMap, która zawiera wszystkie pary klucz-wartość z oryginalnej mapy, ale z odwróconą kolejnością kluczy.

Wskazówka: wykorzystaj `Collections.reverseOrder()`.

L3. Napisz metodę `calculateAverageValue`, która przyjmuje generyczną TreeMap (`TreeMap<K, Double> map`). Metoda powinna obliczać i zwracać średnią wartość wszystkich wartości (typu Double) przechowywanych w mapie. Zakładamy, że mapa nie jest pusta.

26.13 `Vector`

M1. Stwórz funkcję `concatenateVectors`, która przyjmuje dwa obiekty `Vector<T>` i zwraca nowy `Vector<T>`, zawierający wszystkie elementy z pierwszego wektora, a po nich wszystkie elementy z drugiego wektora.

M2. Napisz metodę `reverseVector`, która przyjmuje obiekt `Vector<T>` i zwraca nowy `Vector<T>`, w którym kolejność elementów jest odwrócona względem oryginalnego wektora.

M3. Zaimplementuj funkcję `filterVector`, która przyjmuje `Vector<T>` i interfejs funkcyjny `Predicate<T>`. Funkcja powinna zwracać nowy `Vector<T>`, zawierający tylko te elementy z oryginalnego wektora, które spełniają warunek zdefiniowany przez `Predicate<T>`.

27 Delegacje

1. Wykonaj poniższe czynności:

a. Stwórz interfejs `Printer` z metodą `drukuj(String tekst)`.
b. Utwórz klasę `StandardowyPrinter`, która implementuje `Printer` i wypisuje tekst na konsolę.
c. Utwórz klasę `Biuro`, która posiada prywatne pole typu `Printer`. W konstruktorze `Biuro` przyjmij `Printer` jako argument i przypisz go do pola. Dodaj metodę `drukujDokument(String tekst)`, która będzie delegować zadanie drukowania do obiektu klasy `Printer`.

2. Wykonaj poniższe czynności:

a. Stwórz interfejs `Silnik` z metodami `uruchom()` i `zatrzymaj()`.
b. Utwórz klasę `BenzynowySilnik`, która implementuje `Silnik` i symuluje działanie silnika na benzynę.
c. Utwórz klasę `Samochód`, która posiada prywatne pole typu `Silnik`. W konstruktorze przyjmij `Silnik` jako argument. Dodaj metody `start()` i `stop()`, które będą delegować odpowiednio zadanie uruchomienia i zatrzymania silnika do obiektu klasy `Silnik`.

3. Wykonaj poniższe czynności:

a. Stwórz interfejs `Powiadomienie` z metodą `wyślij(String wiadomość)`.
b. Utwórz klasę `Email`, która implementuje `Powiadomienie` i symuluje wysyłanie wiadomości e-mail.
c. Utwórz klasę `Użytkownik`, która posiada prywatne pole typu `Powiadomienie`. W konstruktorze przyjmij `Powiadomienie` jako argument. Dodaj metodę `powiadomOModernizacji(String informacja)`, która będzie delegować zadanie wysyłania powiadomienia do obiektu klasy `Powiadomienie`.

Część III

Zadania różne

Część IV

Wzorce projektowe

Bibliografia i inne zbiory zadań