Zbiór zadań - Java

Piotr Jastrzębski

2023-07-11

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.

Cześć I

Wprowadzenie do języka Java

1 Instrukcje 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 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 (delta $>$ 0, delta $=$ 0, delta $<$ 0).

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.

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 oblicza sumę cyfr dowolnej wprowadzonej liczby. Program powinien akceptować liczbę jako input od użytkownika.
8. Napisz program, który generuje i wyświetla pierwsze 10 elementów szeregu geometrycznego o zadanym pierwszym elemencie i ilorazie. Parametry szeregu powinny być wprowadzane przez użytkownika.
9. 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.
10. Napisz program, który obliczy sumę kwadratów liczb od 1 do n, gdzie n jest liczbą wprowadzoną przez użytkownika.
11. 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 $1 + 2 + 3 = 6$.
12. 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 $1\hat{}3 + 5\hat{}3 + 3\hat{}3 = 153$.
13. Napisz program, który dla dwóch podanych liczb obliczy ich najmniejszą wspólną wielokrotność (NWW). Użytkownik powinien podać dwie liczby jako dane wejściowe.

Cześć II

Programowanie obiektowe

4 Pojęcie klasy/obiektu

1. Utwórz klasę `Pies` z polami: `imie`, `rasa` i `wiek`. Napisz metodę `szczekaj()`, która wydrukuje na konsoli `"Hau Hau"`. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.
2. Stwórz klasę `Samochod` z polami: `marka`, `model` i `predkosc`. Napisz metody `przyspiesz(int wartosc)` i `zwolnij(int wartosc)`, które odpowiednio zwiększają i zmniejszają prędkość o podaną wartość. Stwórz przypadek testowy, aby wywołać każdą metodę co najmniej jeden raz.
3. Stwórz klasę `KontoBankowe` z polem `saldo`. Napisz metody `wplata(double kwota)` i `wyplata(double kwota)`, które odpowiednio zwiększają i zmniejszają saldo o daną kwotę. Stwórz przypadek testowy, aby wywołać każdą metodę co najmniej jeden raz.
4. Utwórz klasę `Punkt` z dwoma polami: `x` i `y` reprezentującymi współrzędne na płaszczyźnie. Napisz metodę `odleglosc(Punkt innyPunkt)`, która oblicza odległość między bieżącym punktem a innym punktem. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.
5. Stwórz klasę `Czas` z polami: `godziny` i `minuty`. Napisz metodę `dodajCzas(Czas innyCzas)`, która dodaje do bieżącego czasu czas podany jako argument i zwraca nowy obiekt klasy Czas. Zadbaj o to, aby minuty i godziny nie przekraczały odpowiednio 59 i 23. Stwórz przypadek testowy, aby wywołać metodę co najmniej jeden raz.

5 Modyfikatory dostępu

1. Utwórz klasę `Osoba` z publicznym polem `imie` oraz prywatnym polem `haslo`. Zobacz jak różne modyfikatory dostępu wpływają na dostęp do tych pól z innej klasy.
2. Stwórz dwie klasy: `Rodzic` i `Dziecko`. Klasa `Rodzic` powinna mieć jedno pole `protected`. Spróbuj uzyskać dostęp do tego pola z klasy `Dziecko`.
3. Utwórz klasę `Samochod` z prywatną metodą `zawalSilnik()`. Spróbuj wywołać tę metodę z zewnątrz klasy.
4. Stwórz dwie klasy w tym samym pakiecie: `Pracownik` i `Firma`. Klasa `Pracownik` powinna mieć pole bez modyfikatora dostępu. Spróbuj uzyskać dostęp do tego pola z klasy `Firma`.
5. Utwórz klasę `KontoBankowe` z publicznym polem `numerKonta` i prywatnym polem `saldo`. Zobacz, jak różne modyfikatory dostępu wpływają na dostęp do tych pól z innej klasy.

6 Konstruktor

1. Stwórz klasę `Samochod` zawierającą prywatne pola: `marka`, `model`, `rokProdukcji`, `przebieg` oraz `kolor`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) dla wszystkich pól. Następnie dodaj metodę `wyswietlInformacje()`, która wyświetla wszystkie informacje o samochodzie.

2. Stwórz klasę `Osoba` z prywatnymi polami: `imie`, `nazwisko`, `wiek`, `adres`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `przedstawSie()`, która zwraca łańcuch znaków z informacjami o osobie.

3. Stwórz klasę `Ksiazka` z prywatnymi polami: `tytul`, `autor`, `rokWydania`, `wydawnictwo` oraz `liczbaStron`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `pokazInformacje()`, która wyświetla informacje o książce.

4. Stwórz klasę `Punkt2D` 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 `odleglosc(Punkt2D innyPunkt)`, która oblicza odległość między dwoma punktami na płaszczyźnie.

5. Stwórz klasę `Prostokat` z prywatnymi polami `szerokosc` i `wysokosc`. Dodaj konstruktor, który przyjmuje długości boków jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `pole()` i `obwod()`, które obliczają pole powierzchni i obwód prostokąta.

6. Stwórz klasę `Kolo` z prywatnym polem `promien`. Dodaj konstruktor, który przyjmuje promień jako argument. Dodaj metody dostępowe (gettery i settery) oraz metody `pole()` i `obwod()`, które obliczają pole powierzchni i obwód koła.

7. Stwórz klasę `Student` z prywatnymi polami: `imie`, `nazwisko`, `numerIndeksu`, `rokStudiow` oraz `sredniaOcen`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `pokazInformacje()`, która wyświetla informacje o studencie.

8. Stwórz klasę `Pracownik` z prywatnymi polami: `imie`, `nazwisko`, `stanowisko`, `wiek` oraz `placa`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `pokazInformacje()`, która wyświetla informacje o pracowniku.

9. Stwórz klasę `KontoBankowe` z prywatnymi polami: `numerKonta`, `wlasciciel`, `saldo` oraz `typKonta`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `wplac(double kwota)` i `wyplac(double kwota)`, które odpowiednio dodają lub odejmują kwotę od salda konta.

10. Stwórz klasę `Telewizor` z prywatnymi polami: `marka`, `przekatnaEkranu`, `rozdzielczosc`, `czySmartTV` oraz `cena`. Dodaj konstruktor, który przyjmuje wszystkie pola jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `pokazInformacje()`, która wyświetla informacje o telewizorze.

11. Stwórz klasę `DziennikOcen` z prywatnymi polami: `imie`, `nazwisko` oraz `oceny` (jako `ArrayList` typu `int`). Dodaj konstruktor, który przyjmuje imię i nazwisko jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metody `dodajOcene(int ocena)` i `usunOcene(int indeks)`, które odpowiednio dodają lub usuwają ocenę z listy ocen. Dodaj również metodę `sredniaOcen()` do obliczania średniej ocen.

12. Stwórz klasę `HistoriaTemperatur` z prywatnym polem temperatury (jako `ArrayList` typu `double`). Dodaj konstruktor domyślny. Dodaj metody dostępowe (gettery i settery) oraz metody `dodajTemperature(double temperatura)` i `usunTemperature(int indeks)`, które odpowiednio dodają lub usuwają temperaturę z listy temperatur. Dodaj również metodę `sredniaTemperatur()` do obliczania średniej temperatur.

13. Stwórz klasę `WynikiTestow` z prywatnymi polami: `imie`, `nazwisko` oraz `wyniki` (jako tablica typu `int`). Dodaj konstruktor, który przyjmuje imię, nazwisko oraz rozmiar tablicy jako argumenty. Dodaj metody dostępowe (gettery i settery) oraz metodę `dodajWynik(int indeks, int wynik)`, która dodaje wynik testu na podanym indeksie. Dodaj również metodę `sredniWynik()` do obliczania średniego wyniku.

14. Stwórz klasę `ZarzadcaZadan` z prywatnym polem `priorytetyZadan` (jako `ArrayList` typu `int`). Dodaj konstruktor domyślny. Dodaj metody dostępowe (gettery i settery) oraz metody `dodajPriorytet(int priorytet)` i `usunPriorytet(int indeks)`, które odpowiednio dodają lub usuwają priorytet z listy priorytetów. Dodaj również metodę `najwyzszyPriorytet()` do znajdowania najwyższego priorytetu.

15. Stwórz klasę `Magazyn` z prywatnym polem `iloscProduktow` (jako tablica typu `int`). Dodaj konstruktor, który przyjmuje rozmiar tablicy jako argument. Dodaj metody dostępowe (gettery i settery) oraz metodę `dodajProdukty(int indeks, int ilosc)`, która dodaje określoną ilość produktów na podanym indeksie. Dodaj również metodę `sumaProduktow()` do obliczania sumy wszystkich produktów w magazynie.

7 Złożone pola w klasie

1. Utwórz klasę `AlbumMuzyczny` z polami `tytul`, `artysta` oraz `oceny` (jako tablica z elementami typu double). Dodaj metodę pozwalającą na dodawanie i usuwanie ocen. Utwórz klasę `AlbumRockowy`, która dziedziczy po klasie `AlbumMuzyczny`.Klasa `AlbumRockowy` powinna mieć dodatkowe pole `gatunekRocka`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

2. Utwórz klasę `Ksiazka` z polami `tytul`, `autor` oraz `recenzje` (jako tablica z elementami typu double). Dodaj metody pozwalające na dodawanie i usuwanie recenzji. Utwórz klasę `KsiazkaFantasy`, która dziedziczy po klasie `Ksiazka`. Klasa `KsiazkaFantasy` powinna mieć dodatkowe pole `podgatunekFantasy`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

3. Utwórz klasę `GraKomputerowa` z polami `tytul`, `producent` oraz `oceny` (jako tablica z elementami typu double). Dodaj metody pozwalające na dodawanie i usuwanie ocen. Utwórz klasę `GraRPG`, która dziedziczy po klasie `GraKomputerowa`. Klasa `GraRPG` powinna mieć dodatkowe pole `światGry`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

4. Utwórz klasę `Uniwersytet` z polami `nazwa`, `lokalizacja` oraz `kierunkiStudiow` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie kierunków studiów. Utwórz klasę `UniwersytetTechniczny`, która dziedziczy po klasie `Uniwersytet`. Klasa `UniwersytetTechniczny` powinna mieć dodatkowe pole `liczbaLaboratoriow`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

5. Utwórz klasę `GaleriaSztuki` z polami `nazwa`, `miasto` oraz `obrazy` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie obrazów. Utwórz klasę `GaleriaWspolczesna`, która dziedziczy po klasie `GaleriaSztuki`. Klasa `GaleriaWspolczesna` powinna mieć dodatkowe pole `liczbaInstalacji`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

6. Utwórz klasę `Samochod` z polami `marka`, `model` oraz `wariantySilnikow` (jako tablica z elementami typu `String`). Dodaj metody pozwalające na dodawanie i usuwanie wariantów silników. Utwórz klasę `SamochodElektryczny`, która dziedziczy po klasie `Samochod`. Klasa `SamochodElektryczny` powinna mieć dodatkowe pole `zasieg`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

8 Dziedziczenie

1. Utwórz klasę `Pojazd` z polami `marka`, `model` i `rokProdukcji`. Utwórz klasy `Samochod` i `Motocykl`, które dziedziczą po klasie `Pojazd`. Klasa `Samochod` powinna mieć dodatkowe pole `liczbaDrzwi`, a klasa `Motocykl` pole `pojemnoscSilnika`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

2. Utwórz klasę `Pracownik` z polami `imie`, `nazwisko` i `placa`. Utwórz klasy `Programista` i `Tester`, które dziedziczą po klasie `Pracownik`. Klasa `Programista` powinna mieć dodatkowe pole `jezykProgramowania`, a klasa `Tester` pole `typTestowania`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

3. Utwórz klasę `Nieruchomosc` z polami `adres`, `metraż` i `cena`. Utwórz klasy `Dom` i `Mieszkanie`, które dziedziczą po klasie `Nieruchomosc`. Klasa `Dom` powinna mieć dodatkowe pole `liczbaPieter`, a klasa `Mieszkanie` pole `numerPietra`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

4. Utwórz klasę `GraPlanszowa` z polami `nazwaGry`, `minLiczbaGraczy`, `maxLiczbaGraczy` oraz `zasadyGry` (jako ArrayList typu `String`). Utwórz klasy `GraEdukacyjna` i `GraStrategiczna`, które dziedziczą po klasie `GraPlanszowa`. Klasa `GraEdukacyjna` powinna mieć dodatkowe pole `przedmiot`, a klasa `GraStrategiczna` pole `czasTrwania`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

5. Utwórz klasę `Druzyna` z polami `nazwa`, `miasto` oraz `punkty` (jako ArrayList typu `Integer`). Utwórz klasy `DruzynaPilkarska` i `DruzynaSiatkarska`, które dziedziczą po klasie `Druzyna`. Klasa `DruzynaPilkarska` powinna mieć dodatkowe pole `pozycjaWRankingu`, a klasa `DruzynaSiatkarska` pole `liczbaZwyciestw`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

6. Utwórz klasę `Komputer` z polami `producent`, `model` oraz `cenyCzesci` (jako ArrayList typu `Double`). Utwórz klasy `Laptop` i `Stacjonarny`, które dziedziczą po klasie `Komputer`. Klasa `Laptop` powinna mieć dodatkowe pole `waga`, a klasa `Stacjonarny` pole `obudowa`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

7. Utwórz klasę `AlbumMuzyczny` z polami `tytul`, `artysta` oraz `oceny` (jako ArrayList typu `Integer`). Utwórz klasy `AlbumRockowy` i `AlbumJazzowy`, które dziedziczą po klasie `AlbumMuzyczny`. Klasa `AlbumRockowy` powinna mieć dodatkowe pole `gatunekRocka`, a klasa `AlbumJazzowy` pole `gatunekJazzu`. Dodaj konstruktory, metody gettery i settery, metodę `toString()` oraz `equals()` dla każdej z klas. Napisz program testujący zdefiniowane klasy i metody.

9 Klasy abstrakcyjne

1. Zdefiniuj abstrakcyjną klasę `NarzedziePracy` z polami `nazwa` typu `String` oraz `rokProdukcji` typu `java.time.LocalDate`. Dodaj metodę abstrakcyjną `uzyj()`, która będzie symulować użycie narzędzia. Następnie zdefiniuj klasy `Mlotek`, `Srubokret` i `Pila`, które dziedziczą po klasie `NarzedziePracy` i implementują metodę `uzyj()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.
2. Zdefiniuj abstrakcyjną klasę `GrafikaKomputerowa` z polami `szerokosc`, `wysokosc` typu `int` oraz `nazwaPliku` typu `String`. Dodaj abstrakcyjne metody `wczytajPlik()` i `zapiszPlik()`. Następnie zdefiniuj klasy `Bitmapa` i `Wektor`, które dziedziczą po klasie `GrafikaKomputerowa` i implementują metody `wczytajPlik()` oraz `zapiszPlik()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.
3. Zdefiniuj abstrakcyjną klasę `UrządzenieElektroniczne` z polami `producent` typu `String`, `model` typu `String` oraz `rokProdukcji` typu `java.time.LocalDate`. Dodaj abstrakcyjne metody `włącz()` i `wyłącz()`. Następnie zdefiniuj klasy `Smartfon`, `Telewizor` i `Laptop`, które dziedziczą po klasie `UrządzenieElektroniczne` i implementują metody `włącz()` oraz `wyłącz()`. Stwórz listę tablicową odpowiednich 5 obiektów i wywołaj dla nich napisaną metodę.

10 Interfejs `Comparable`

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

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

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

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

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

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

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

8. Napisz klasę `ProduktSpozywczy`, która zawiera pola: `nazwa` (typu `String`), `cena` (typu `double`) i `dataWaznosci` (typu `LocalDate`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `ProduktSpozywczy` były sortowane rosnąco według daty

ważności. Stwórz tablicę 4 obiektów klasy `ProduktSpozywczy` i posortuj ją według sprecyzowanego kryterium.

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

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

11. Napisz klasę `Student`, która zawiera pola: `imie` (typu `String`), `sredniaOcen` (typu `double`) i `rokStudiow` (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 `Zamowienie`, która zawiera pola: `nazwaProduktu` (typu `String`), `ilosc` (typu `int`) i `cenaJednostkowa` (typu `double`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Zamowienie` były sortowane według jednego kryterium: malejąco według ceny jednostkowej, a przy równości sortowane były rosnąco według ilości. Stwórz listę tablicową 4 obiektów klasy `Zamowienie` i posortuj ją według sprecyzowanego kryterium.

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

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

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

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

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

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

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

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

21. Zdefiniuj klasę Klient, która będzie implementować generyczny interfejs Comparable. W klasie tej zadeklaruj prywatne pola nazwisko typu String oraz saldo 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 Firma dziedziczącą po klasie Klient. Klasa Firma ma dodatkowo posiadać prywatne pole liczbaPracowników typu int. Implementując metodę compareTo interfejsu Comparable w klasie Firma, skorzystaj z metody compareTo zdefiniowanej w klasie Klient oraz, w razie potrzeby, uwzględnij pole liczbaPracowników. Napisz program TestKlient, w którym utwórz listę 5 klientów i firm o nazwie listaKlientów 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 listaKlientów, posortuj ją za pomocą instancyjnej metody sort z klasy ArrayList i ponownie wyświetl zawartość tej listy.

22. Zdefiniuj klasę Zwierzę, która będzie implementować generyczny interfejs Comparable. W klasie tej zadeklaruj prywatne pola gatunek typu String oraz wiek 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 Pies dziedziczącą po klasie Zwierzę. Klasa Pies ma dodatkowo posiadać prywatne pole rasa typu String. Implementując metodę compareTo interfejsu Comparable w klasie Pies, skorzystaj z metody compareTo zdefiniowanej w klasie Zwierzę oraz, w razie potrzeby, uwzględnij pole rasa. Napisz program TestZwierzę, w którym utwórz listę 5 zwierząt i psów o nazwie listaZwierząt posługując się klasą ArrayList. W składzie listy powinny wystąpić przynajmniej

11 Interfejs `Comparator`

1. Napisz klasę Osoba z polami imię (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 Samochód 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 imię (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 Książka z polami tytuł (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 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 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 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 następnie 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 po identyfikatorze).

12 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.

Cześć III

Zadania różne

Cześć IV

Wzorce projektowe

Bibliografia i inne zbiory zadań

Kozak, Jan. b.d. *Materiały do ćwiczeń*. http://www.jkozak.pl/przedmioty/podstawy-i-jezyki-programowania/materialy-do-cwiczen/.

Rychlicki, Wiesław. 2012. *Programowanie w języku Java. Zbiór zadań z (p)odpowiedziami*. Helion.