

Nauka programowania i język Java – ćwiczenia do domu

0.

Rozwiązujcie i przysyłajcie na adres p.czarnik@alx.pl

1. Zmienne, if-y i pierwsze pętle

Zadanie 1.1 *Interakcja i proste obliczenia*

Napisz program, który prosi użytkownika (przez `JOptionPane.showInputDialog`), żeby podał, ile kosztuje kilo ziemniaków. Niech program policzy i wyświetli, ile trzeba będzie zapłacić za pięć kilo ziemniaków.

Potem napisz program, który prosi użytkownika (przez `JOptionPane.showInputDialog`), żeby podał, ile kosztuje kilo ziemniaków i ile kilo chce kupić. Niech program policzy i wyświetli, ile trzeba będzie zapłacić za te ziemniaki.

Potem napisz program, który prosi użytkownika (przez `JOptionPane.showInputDialog`), żeby podał, ile kosztuje kilo ziemniaków, ile kilo ziemniaków chce kupić, ile kosztuje kilo bananów i ile kilo bananów chce kupić. Niech program policzy i wyświetli, ile trzeba będzie zapłacić za te ziemniaki i banany razem. I niech program sprawdzi i powie, za co trzeba będzie zapłacić więcej - za banany czy za ziemniaki.

Zadanie 1.2 *Pole trójkąta*

Program, który odczytuje trzy liczby, sprawdza czy liczby te mogą stanowić boki trójkąta (np. z 2, 2 i 5 nie da się ułożyć trójkąta, prawa?), a jeśli mogą, oblicza pole powierzchni trójkąta o takich bokach.

Wzór Herona: $\sqrt{p(p-a)(p-b)(p-c)}$, gdzie p jest połową obwodu: $(a+b+c)/2$.

Tutaj użyj jednego z poznanych sposobów komunikacji z użytkownikiem. Pierwiastek kwadratowy to metoda `Math.sqrt()`.

Zadanie 1.3 *Szewc*

Napisz taki program: użytkownik ma podać, w jaki dzień tygodnia oddał buty do szewca (numer od 1 do 7). Ma też podać, ile dni będzie trwała naprawa. Program ma wypisać, w jaki dzień tygodnia buty będą gotowe do odbioru. W podstawowej wersji możesz wypisywać dzień odbioru też jako numer. Postaraj się obsłużyć także sytuację, że naprawa trwa dłużej niż 7 dni. Na końcu zrób wersję, w której program wypisuje dzień odbioru słownie.

Zadanie 1.4 Firma remontowa

Firma remontowa posiada taki cennik usług (nie mam pojęcia czy realistyczny ;)):

- gipsowanie ścian: 100 zł za metr kwadratowy ściany
- malowanie ścian i sufitów: 30 zł za metr kwadratowy
- położenie paneli podłogowych: 50 zł za metr kwadratowy podłogi
- położenie listew przypodłogowych: 40 zł za metr bieżący

Napisz program, który pomaga wycenić pracę na podstawie wymiarów pomieszczenia. Zakładając, że pomieszczenie ma kształt prostokąta, program powinien zapytać o dwa wymiary poziome (w metrach) oraz o wysokość i na tej podstawie obliczyć powierzchnię podłogi, sufitu oraz łącznie wszystkich ścian, a także obwód pokoju (listy podłogowe).

Wersja 1: Przyjmij, że zawsze wykonywany jest komplet robót: gipsowanie ścian, malowanie ścian i sufitu, panele podłogowe, listwy przypodłogowe. Program na podstawie danych wejściowych oblicza sumaryczny koszt prac.

Wersja 2: Zapytaj użytkownika jakie elementy prac są wykonywane. Jedną z opcji jest skorzystanie z `JOptionPane.showConfirmDialog` – popatrz na moje proste przykłady, przeczytaj dokumentację, poszukaj wskazówek w internecie.

Jako domyślną wysokość pomieszczenia można wpisać 2.50, ale tak, aby użytkownik mógł zmienić. Można to zrobić tak: `JOptionPane.showInputDialog("Podaj wysokość", "2.50");`

2. Funkcje, pętle, tablice...

Zadanie 2.1 Zgadnij liczbę z zakresu

```
Random r = new Random(); int x = r.nextInt(1000);
```

Program losuje liczbę z zakresu od 0 do 999 (jak wyżej). Użytkownik ma zgadnąć tę liczbę nie widząc jej. Kiedy użytkownik poda nieprawidłowy wynik, program podpowiada pisząc czy podana liczba była za duża, czy za mała. Gdy użytkownik poda właściwą liczbę, program wypisuje gratulacje jednocześnie informując, w której próbie udało się zgadnąć liczbę.

Nawiasem mówiąc technika wyszukiwania oparta o „podpowiedzi” *za dużo/za mało* nazywa się **bisekcją** i pełni w informatyce bardzo ważną rolę. Umiejętnie ją stosując powinno się te zagadki rozwiązywać w 9-10 próbach (bo $2^{10} = 1024$).

Zadanie 2.2 Podzielne przez 3 lub przez 5

Użytkownik podaje liczbę całkowitą „limit”. Następnie program wypisuje, bez powtórzeń, liczby z zakresu od 1 do podanego limitu włącznie, które są podzielne przez 3 lub przez 5. Wypisz także jak dużo takich liczb wystąpiło w tym przedziale.

Przykładowe działanie programu:

Podaj limit: 20

3 5 6 9 10 12 15 18 20

Takich liczb było: 9

Zadanie 2.3 Choinka

Napisz program, który wczytuje liczbę całkowitą, a następnie na konsolę wypisuje w tylu liniach „choinkę” ze znaków *. Np. dla parametru 3 powinno się wypisać:

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

Podpowiedź: `System.out.print(...)` wypisuje i nie przechodzi do nowej linii, `System.out.println()` przechodzi do nowej linii.

Zadanie 2.4 Tabliczka mnożenia

Napisz program, który wypisuje na ekran tradycyjną tabliczkę mnożenia 10×10.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Rozszerzenie dla chętnych: niech program pyta użytkownika o rozmiar tabliczki (inną liczbę zamiast 10).

Zadanie 2.5 Funkcje na liczbach

Napisz funkcje:

1. `int sumaCyfr(long n)`
Zwraca sumę wartości cyfr, z których składa się liczba `n`. Np. dla parametru 1023 wynikiem powinno być 6.
2. `long potega(long podstawa, int wykladnik)`
Potęgowanie liczb całkowitych (za pomocą mnożenia). Ma działać dla wykładnik ≥ 0 .
3. `long fib(int n)`
Zwraca `n`-tą liczbę Fibonacciego, gdzie liczby Fibonacciego są zdefiniowane następująco:
`fib(0) = 0`
`fib(1) = 1`
`fib(n) = fib(n-2) + fib(n-1)`
Co oznacza, że początkowe liczby Fibonacciego to: 0 1 1 2 3 5 8 13 21 34
Postaraj się do następnego spotkania ;) obliczyć osiemdziesiątą liczbę Fibonacciego (jeszcze mieści się w `long`).
Dla chętnych: wersja `BigInteger` – bez ograniczenia na wielkość wyniku.
4. `boolean czyPierwsza(long liczba)`
Sprawdza czy liczba jest pierwsza i zwraca **true** albo **false**.

Sprawdź czy funkcje działają prawidłowo pisząc program lub programy, w których są uruchamiane.

Zadanie 2.6 Funkcje na tablicach liczb

Każde z tych ćwiczeń należy zrealizować jako funkcję (metodę statyczną), która otrzymuje w parametrze tablicę (`int[]` dla liczb całkowitych). Niektóre z metod będą wymagały podania dodatkowych parametrów.

Pomijając te, które zrobiliśmy na zajęciach, ewentualnie jeszcze raz jako powtórzenie.

- `void wypiszPodzielne(int[] tab, int x)` – wypisuje na `System.out` wszystkie te liczby z tablicy `tab`, które są podzielne przez `x` (warunek do sprawdzenia: `element % x == 0`)
- `Integer pierwszaPodzielna(int[] tab, int x)` – zwraca (return) pierwszą znaną w `tab` liczbę podzielną przez `x`; zwraca `null`, jeśli takiej liczby tam nie ma.
- `int sumaPodzielnych(int[] tab, int x)` – liczy sumę tych elementów tablicy, które są podzielne przez `x`.
- `int ilePodzielnych(int[] tab, int x)` – liczy ile elementów tablicy `tab` jest podzielnych przez `x`.
- `double sredniaPodzielnych(int[] tab, int x)` – liczy średnią arytmetyczną tych elementów tablicy, które są podzielne przez `x`.
- `Integer max(int[] tab)` – zwraca najmniejszą wartość z tablicy
 - W przypadku pustej tablicy można zwrócić `null`
- `Integer min(int[] tab)` – zwraca najmniejszą wartość z tablicy
- `int roznicaMinMax(int[] tab)` – różnica pomiędzy największą a najmniejszą liczbą w tablicy; 0 jeśli tablica jest pusta.
- `Integer znajdzWspolny(int[] t1, int[] t2)` – zwraca element (liczbę), który występuje zarówno w tablicy `t1`, jak i `t2`; zwraca `null`, jeśli takiego elementu nie ma.
- `List<Integer> wszystkieWspolne(int[] t1, int[] t2)` – zwraca listę wszystkich wspólnych elementów z tablic `t1` i `t2`. Jeśli takiego elementu nie ma, należy zwrócić pustą listę. (dla tych, którzy znają listy lub chcą poszukać jak się ich używa).

Zadanie 2.7 Skarb

Napisz grę tekstową polegającą na poszukiwaniu skarbu na dwuwymiarowej planszy o rozmiarach 10 na 10. Program na początku losuje pozycję skarbu oraz pozycję gracza.

Następnie użytkownik może wprowadzać komendy zmieniające położenie postaci o jedną pozycję w górę/dół/lewo/prawo (np zgodnie z konwencją WSAD) – normalnie za pomocą Scannera.

Gdy gracz wejdzie na pole, na którym kryje się skarb – wygrywa.

Gdy wyjdzie poza planszę – przegrywa.

Po każdym ruchu użytkownik powinien otrzymywać informację o tym, czy zmierza w dobrym kierunku (zbliżasz się / oddalasz się). Po znalezieniu skarbu wypisz liczbę ruchów wykorzystanych przez użytkownika na dojście do celu.

Zadanie 2.8 Kalkulator konsolowy

Napisz program działający w konsoli, który na podstawie dwóch podanych liczb oraz znaku operacji obliczy wynik działania matematycznego. W przypadku podania nieprawidłowej operacji lub złego formatu liczb program ma wyświetlić komunikat o błędzie.

Obsłuż co najmniej cztery podstawowe działania matematyczne (+ - * /), dodatkowo możesz inne (sprawdź możliwości klasy Math).

Przykładowa sesja programu (między liczbami a znakiem mnożenia są spacje!) :

Podaj działanie: 12 * 3

Wynik: 36

Najlepiej, aby program działał w pętli i wielokrotnie pytał o działanie. Sami opracujcie sposób kończenia programu, jest kilka możliwości...

Zadanie 2.9 Range

Jedną z często używanych możliwości języka Python jest łatwe przechodzenie przez zakresy liczbowe za pomocą funkcji `range` oraz wyrażeń listowych. Jeśli masz dostęp do interpretera Pythona, spróbuj np. następujących wyrażeń:

```
list(range(10))      list(range(5, 10))      list(range(10, 20, 4)) list(range(20, 10, -3))
```

Zaimplementuj podobną funkcjonalność w Javie jako program, który przyjmuje parametry wiersza poleceń (`String[] args` w `main`) i wypisuje ciąg liczb rozdzielonych spacją na standardowe wyjście.

Program przyjmuje od 1 do 3 parametrów wiersza poleceń – liczb całkowitych. Ich ilość jest dostępna jako `args.length`, a kolejne parametry jako `args[0]`, `args[1]` i `args[2]`.

- Pierwszy parametr, o ile podano co najmniej dwa, oznacza początek zakresu. Jeśli podano tylko jeden parametr, to domyślnym początkiem zakresu jest 0.
- Drugi parametr, a jedyny w przypadku gdy podano tylko jeden, oznacza koniec zakresu, czyli liczbę, przed którą program ma zakończyć wypisywanie.
- Trzeci parametr oznacza wartość kroku, o jaki różnią się kolejne liczby ciągu. Domyślną wartością kroku jest 1. Krok może być ujemny.

Ten program możesz spróbować napisać w zwykłym edytorze (typu Notepad++, a nie Eclipse/IntelliJ). Program najłatwiej przetestować w konsoli. Przykładowe wywołania i ich wyniki (zakładają, że klasa nie należy do żadnego pakietu - jak nasze przykłady z pierwszego dnia):

```
javac Range.java
java Range 10
0 1 2 3 4 5 6 7 8 9

java Range 0

java Range 5 10
5 6 7 8 9

java Range 10 20 4
10 14 18

java Range 20 10 -3
20 17 14 11
```


3. Klasy

Zadanie 3.1 Klasy dla ogłoszeń

Zaproponuj klasy, za pomocą których będzie się zapisywać ogłoszenia (takie jak w serwisie internetowym z ogłoszeniami). Użyj właściwych typów dla poszczególnych pól.

Najlepiej, aby klasa **Ogłoszenie** opisywała rzeczy, które posiada każde ogłoszenie, m.in. tytuł, opis, cenę, dane kontaktowe sprzedawcy, a dodatkowo stwórz klasy, które rozszerzają tę klasę („podklasy”) opisujące konkretne rodzaje ogłoszeń.

OgłoszenieSamochodowe – dziedziczy z **Ogłoszenie** i dodatkowo określa cechy sprzedawanego samochodu jak model, markę, rok produkcji, stan licznika, pojemność, moc i rodzaj paliwa (spróbuj użyć **enum**).

OgłoszenieMieszkaniowe – też dziedziczy z **Ogłoszenie**, a dodaje cechy sprzedawanego mieszkania / domu: miejscowość, metraż, liczba pokoi. Ewentualnie można utworzyć kolejne poziomy podklas: osobno dla nieruchomości różnego typu (mieszkanie / dom).

Własne pomysły mile widziane.

W używanym IDE wygeneruj standardowe konstruktory i metody. Dodaj inne metody według uznania. Napisz program, który tworzy i wypisuje kilka takich obiektów.

Zadanie 3.2 Klasa dla ułamków

Stwórz klasę, której obiekty reprezentują liczby wymierne w postaci ułamkowej. Klasa powinna implementować operacje na ułamkach.

Z kilku możliwych podejść najbardziej rekomenduję utworzenie klasy niemutowalnej, zgodnie z wzorcem „value object”: pola final, brak setterów, operacje matematyczne zawsze zwracają nowy obiekt w wyniku, a nie modyfikują bieżącego; metody porównujące biorą pod uwagę wartość, a nie tożsamość obiektu. Przykładami takich klas są BigDecimal oraz LocalDate.

Nie dopuszczaj do sytuacji, aby w mianowniku znalazło się zero – wyrzucaj wyjątek IllegalArgumentException. Zalecam taką „normalizację” tworzonych ułamków, aby mianownik był zawsze dodatni, a tylko licznik mógł być ujemny. Operacje arytmetyczne powinny także zwracać wynik w postaci maksymalnie skróconej.

Proponowane publiczne API klasy (tym razem piszę po angielsku):

- static Fraction of(long nom, long denom)
- static Fraction of(long number) // na podstawie liczby całkowitej, mianownik = 1
- static final Fraction ZERO, ONE, HALF, ... – kilka stałych, podobnie jak w BigInteger
- long getNominator(), long getDenominator()
- String toString()
- equals + hashCode – w oparciu o wartości licznika i mianownika,
imo $\frac{3}{4}$ i $\frac{6}{8}$ powinny być uznane za różne, podobnie jak jest to w klasie BigDecimal, gdy wartość jest równa, ale skala (precyzja) się różni...
- int compareTo(Fraction other) – klasa powinna implementować interfejs Comparable
 - Fraction shortened() – zwraca ułamek o tej samej wartości, skrócony w miarę możliwości (o największy wspólny dzielnik licznika i mianownika)
- Fraction reciprocal() – zwraca odwrotność ułamka
- Fraction neg() – zwraca liczbę ze zmienionym znakiem (plus/minus)
- Fraction add(Fraction) – suma
- Fraction sub(Fraction) – różnica
- Fraction mul(Fraction) – iloczyn
- Fraction div(Fraction) – iloraz
- double getAsDouble() – zwraca przybliżoną wartość tego ułamka jako double

4. Pliki i dane

Zadanie 4.1 Pan Tadeusz - policz linie

Użytkownik podaje słowo, a program liczy ile linii w pliku Pan Tadeusz zawiera podane słowo.

Zadanie 4.2 Policz wybrane słowo w całym pliku

Napisz program, który obliczy ile razy dane słowo występuje w pliku (np. Tadeusz w pliku pan-tadeusz.txt).

Podpowiedź:

Najlepiej użyć klasy Scanner bezpośrednio na pliku i użyć odpowiedniego wyrażenia regularnego opisującego separator pomiędzy słowami. Najlepszy sposób to:

```
Scanner sc = new Scanner(new File("pan-tadeusz.txt"));
sc.useDelimiter("[^\\p{L}]+");
while(sc.hasNext()) {
    String slowo = sc.next();
    // TODO co zrobić z każdym słowem
}
```

Wersja interaktywna

Użyj JOptionPane do pobrania szukanego słowa i wyświetlenia wyniku.

Użyj JFileChooser do wybrania pliku z dysku (showOpenDialog i inne kroki - poszukajcie "w internetach" jak się tego używa...).

Zadanie 4.3 Policz wszystkie słowa

Napisz w Javie program, który czyta plik tekstowy i wylicza oraz wypisuje bez powtórzeń wszystkie słowa występujące w pliku wraz z informacją ile razy dane słowo występuje. Na przykład w ten sposób (dla pliku pan-tadeusz.txt):

taczkach	→	1
Tadeusz	→	107
Tadeusza	→	54
Tadeuszek	→	1

W implementacji czytania słów z pliku pomoc może klasa `Scanner` i odpowiednie wyrażenie regularne:

```
Scanner sc = new Scanner(new File("pan-tadeusz.txt"));
sc.usePattern("[^\\p{L}\\d]+");
while(sc.hasNext()) {
    String słowo = sc.next();
    ...
}
```

Rozwiązania można napisać w oparciu o pętlę lub strumieniowo (albo robić dwie wersje).

Dalsze rozszerzenia:

1. Posortuj wypisywane słowa alfabetycznie.
2. W drugiej wersji posortuj według policzonej ilości tych słów w pliku.
3. Po wykonaniu poniższego zadania „Odmiana słów”, policz słowa sprowadzone do swojej formy podstawowej (pierwszej, jeśli jest kilka kandydatur). Przykładowo zamiast całej serii odmienionych Tadeuszów

Tadeusz	→	107
Tadeusza	→	54
Tadeuszem	→	2
Tadeuszowi	→	5
Tadeuszu	→	6

powinien być jeden wpis

Tadeusz → 174

Zadanie 4.4 Odmiana słów

Pod adresem zil.ipipan.waw.pl/PoliMorf znajdują się zasoby dotyczące odmiany wyrazów w języku polskim. Plik `PoliMorf-0.6.7.tab` (lub nowsza wersja) zawiera relację między słowem (pierwsza kolumna) a jego formą podstawową (druga kolumna).

Stwórz klasę, której obiekt zawiera odpowiednio przygotowaną „bazę wiedzy” nt. odmiany słów. Obiekt tej klasy tworzy się na podstawie pliku takiego jak `PoliMorf-0.6.7.tab`. Obiekt powinien umożliwiać znalezienie formy podstawowej (lub kilku kandydatów) dla podanego słowa odmienionego oraz odczytanie zbioru słów odmienionych na podstawie formy podstawowej.

Przykładowe użycie tej klasy mogłoby wyglądać tak (ale nazwy możecie wymyślić inne):

```
BazaOdmiany          baza          =          BazaOdmiany.wczytaj("PoliMorf-0.6.7.tab");
```

```
Set<String> odmiany = baza.znajdzOdmiany("Tadeusz"); // [Tadeusz, Tadeuszowi, ...]
```

```
String podstawowa = baza.znajdzFormePodstawowa("Tadeusza"); // Tadeusz
```

Ten sam wyraz może być formą odmienioną dla kilku form bazowych, np. „dam” może pochodzić od „dać”, ale również od „dama”... Dlatego lepszym, bardziej ogólnym rozwiązaniem byłoby:

```
Set<String> podstawowe = baza.znajdzFormyPodstawowe("Tadeusza"); // [Tadeusz]
```

Korzystając z tej klasy napisz program, który w pętli odczytuje od użytkownika kolejne słowa i dla podanego słowa wyświetla jego formę podstawową lub informację, że nie znaleziono.

Wskazówka nt wydajności: Przyjmijcie, że budowanie bazy wiedzy w pamięci na podstawie pliku może chwilę potrwać (ale raczej sekundy niż minuty) i zająć trochę pamięci, natomiast szukanie form bazowych i odmian ma już działać szybko (ułamki sekund).

Wróć do punktu 3 w rozszerzeniach poprzedniego zadania.

Zadanie 4.5 Dane z pliku "sprzedaż"

W pliku sprzedaz.csv znajdują się dane opisujące sprzedaż różnych sklepów - w każdej linii jest jedna transakcja. Cena jest ceną jednostkową, w osobnej kolumnie jest podana ilość sztuk, a dopiero ich iloczyn jest wartością transakcji.

Utwórz odpowiednie klasy i napisz programy, które czytają dane z tego pliku i obliczają:

1. Sumę wartości wszystkich transakcji z całego pliku.
2. Ilość, sumę oraz minimalną, maksymalną i średnią wartość transakcji z wybranego miasta (niech program pyta o nazwę miasta na początku).
3. Dla każdego miasta występującego w pliku – sumę transakcji z tego miasta (wypisać bez powtórzeń), czyli zastosować schemat grupowania.

5. Aplikacje okienkowe (Swing)

Stwórz jedną lub więcej aplikacji okienkowych w technologii Swing. Wygląd interfejsu przygotuj w edytorze wizualnym, np. w NetBeans (New JFrameForm), ewentualnie w Eclipse z doinstalowaną wtyczką Window Builder (New WindowBuilder > Swing Designer > Application Window) lub w IntelliJ.

Można stworzyć aplikację według własnego pomysłu (mile widziane), albo wybrać coś poniższych propozycji.

Zadanie 5.1 BMI *(tylko, jeśli ktoś potrzebuje bardzo prostego zadania)*

Człowiek podaje swój wzrost i wagę, a otrzymuje wyliczony współczynnik BMI i informację tekstową czy jest w normie, czy ma się odchudzać, czy raczej przytyć.

Niektóre źródła na temat BMI rozróżniają normy ze względu na wiek lub płeć. Opcjonalnie możesz w swoim programie uwzględniać także te informacje.

Zadanie 5.2 Konwerter jednostek

Napisz program, który służy do przeliczania wartości między różnymi systemami miar. Minimum to program, który przelicza jedną parę, np. mile na kilometry. Można spróbować zrobić bardziej rozbudowaną aplikację, np. po jednej stronie ekranu umieścić pola na dane w jednostkach metrycznych (centymetry, metry, kilometry, kilogramy, Celjusz), a z drugiej brytyjskich (cale, stopy, mile, funty, Fahrenheity), a za pomocą przycisków można przeliczać w jedną lub w drugą stronę. Własne pomysły na układ okna i sposób działania mile widziane.

Zadanie 5.3 Automat na monety

Zrealizuj przykład z automatem parkingowym jako aplikację okienkową. Użytkownik podaje liczbę godzin, za które płaci, automat wylicza opłatę, następnie „wrzuca się monety” (np. przyciski dla różnych monet), a automat odejmuje wrzucone monety od kwoty do zapłaty, na końcu „wydaje resztę”.

Zamiast automatu parkingowego można wymyślić np. automat biletowy (z biletami normalnymi i ulgowymi), z kawą, z biletami do ZOO (normalne, ulgowe, rodzinne – z listy do wyboru) itp.

Zadanie 5.4 Kółko i krzyżyk

Program, który pozwala przeprowadzić rozgrywkę w kółko i krzyżyk. Można np. w oknie розміścić 9 przycisków, w których będą odpowiednie symbole. Gdy użytkownik kliknie w przycisk, jest to traktowane jako ruch i symbol się zmienia. Program prawidłowe kliknięcia traktuje na przemian jako ruchy jednego lub drugiego gracza. Program sam powinien rozpoznać kiedy dochodzi do wygranej lub kiedy wszystkie pola zostają zajęte i gra kończy się remisem.

To zadania można spróbować zrobić bez edytora graficznego, pisząc wszystko od zera i używając GridLayout do rozmieszczenia 9 guzików.

Zadanie 5.5 Wilk, koza, kapusta

Starożytna łamigłówka: Po jednej stronie rzeki znajdują się **wilk**, **koza** i **kapusta** oraz przewoźnik, który ma łodzią przewieźć je na drugą stronę rzeki. Problem polega na tym, że w łodzi zmieści się tylko jedno zwierzę/rzecz na raz – przewoźnik musi więc przewozić po jednej rzeczy i zostawiać je na brzegu. Jeśli jednak bez opieki pozostaną wilk i koza – wilk zje kozę; gdy zostaną zaś koza i kapusta – koza zje kapustę. Łamigłówka polega na tym, aby ustalić jak bezpiecznie przewieźć wszystkie trzy elementy na drugą stronę rzeki.

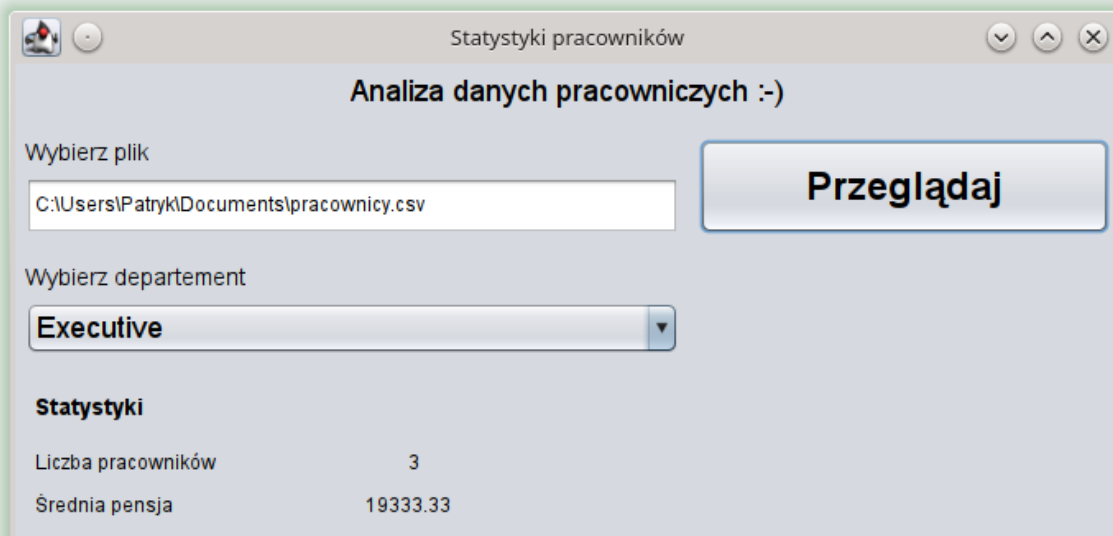
Zgodnie z opisanymi zasadami napisz prostą grę jako program w technologii Swing. W prostszej wersji możesz użyć wyłącznie przycisków i etykiet tekstowych. W miarę możliwości i dostępnego czasu możesz spróbować wprowadzić obrazki (poczytaj np. o ImageIcon i dodawaniu ich do JLabel/JButton).

Zadanie 5.6 Analiza danych z pliku CSV

Program ma umożliwiać analizę danych wczytywanych z pliku takiego jak emps.csv (ewentualnie można zrobić wersję dla zawodnicy.csv lub sprzedaz.csv...).

Po naciśnięciu przycisku "Przeglądaj" za pomocą JFileChooser można wybrać plik z dysku. Używając FileNameExtensionFilter wyświetlaj tylko pliki .csv. Po wybraniu pliku, jego dane powinny zostać wczytane (jako List<Employees>) i w odpowiednich labelach wyświetlone informacje sumaryczne: liczba pracowników, średnia pensja (ew. jeszcze min i max pensja).

Jeśli się uda: W oknie umieść JComboBox, z którego można wybrać nazwę departamentu. Lista departamentów (jako DefaultComboBoxModel) powinna być ustalona na podstawie danych z pliku! Po wybraniu departamentu z listy, w odp. miejscach okna wyświetl informację o ilości pracowników w tym departamencie oraz ich średniej pensji.



6. Zadania/wyzwania na koniec

Zadanie 6.1 Przelicznik walut - wersja webowa

Wykonaj prezentowany na zajęciach okienkowy przelicznik walut jako aplikację webową (za pomocą Springa). Aplikacja ma działać na lokalnym serwerze (localhost), ale pobierać kursy walutowe z serwera NBP tak, jak to robi aplikacja okienkowa. Możesz wykorzystać gotowe klasy ObsługaNBP, Waluta, Tabela...

W pierwszej wersji niech użytkownik wszystkie dane wpisuje w pola tekstowe (<input type='text'>), także kod waluty. Gdy ta wersja zadziała, spróbuj przygotować wersję, gdzie waluty są do wyboru z rozwijanej listy (w HTML-u służą do tego znaczniki <select><option>). Lista walut ma być wyświetlana na podstawie tabeli walut pobranej z serwera.

Opcjonalnie: pozwól na wybór daty z pola input type="date" i obsłuż odpowiednio po stronie Javy (aplikacja dostanie datę jako String). Jeśli data jest podana - aplikacja powinna pobrać kursy archiwalne z tej daty. Jeśli nie podana - powinna użyć kursów aktualnych.

Zadanie 6.2 Liczba słownie

Stwórz klasę (+ ewentualnie klasy pomocnicze w tym samym pakiecie), w której zdefiniowana będzie funkcja zamieniająca podaną liczbę całkowitą na postać słowną. Przykładowe użycie:

```
String slownie = LiczbaSlownie.liczbaSlownie(113);
```

```
// "sto trzynaście"
```

Obsłuż jak największy zakres liczbowy, może cały zakres typu long?

Opcjonalnie dodaj także drugą funkcję zwracającą tekst mówiący o kwocie pieniężnej, tak jak umieszcza się ją np. na umowach czy fakturach, np.:

```
String kwota = LiczbaSlownie.kwotaSlownie(204);
```

```
// "dwieście cztery złote"
```

Opcjonalnie - obsługa „groszy”, wersje w innych językach i inne własne rozszerzenia.

Napisz testy jednostkowe oraz program interaktywny.

Dla chętnych: program okienkowy / prosta aplikacja webowa w Spring MVC udostępniające opisaną funkcjonalność.

Jeśli stworzysz klasę Faktura (wcześniejsze zadanie), to możesz w niej użyć swojego algorytmu do tworzenia kwoty słownie.

Zadanie 6.3 Klasa Faktura

Zdefiniuj klasę służącą do wystawiania faktur. W jej implementacji wykorzystaj kolekcje (listy, mapy itp.). Możesz także stworzyć dodatkowe klasy (np. dla pozycji na fakturze).

Najlepiej utwórz zestaw testów JUnit, które będą sprawdzać, czy klasa działa prawidłowo. Możesz także napisać program konsolowy (dla chętnych i dysponujących czasem: okienkowy), który pozwoli wpisywać dane do faktury, a następnie wypisuje podsumowanie z wszystkimi pozycjami, sumą do zapłaty i podsumowaniem podatków.

Przykładowe operacje, które powinny działać:

```
Faktura faktura = new Faktura("1234/2019");
```

tworzy obiekt reprezentujący fakturę o podanym numerze. Początkowo faktura nie zawiera żadnych pozycji (wpisów).

```
faktura.dodajPozycję("Pralka automatyczna", 2, 1250.00, 23)
```

dodaje pozycję do faktury: nazwa towaru, ilość, cena jednostkowa netto, stawka VAT

```
int ile = faktura.iloscPozycji();
```

zwraca liczbę niepustych pozycji w fakturze.

```
double wartosc1 = faktura.wartoscPozycji(1);
```

zwraca wartość brutto z danej pozycji na fakturze (numeracja od 1).

```
double doZapłaty = faktura.doZapłaty();
```

zwraca sumę do zapłaty za całą fakturę.

```
faktura.wydrukuj();
```

wypisuje na ekran podsumowanie zawierające wszystkie pozycje, a dla każdej z nich:

- nr porządkowy (od 1),
- nazwa,
- cena jednostkowa netto,
- stawka VAT,
- wartość netto,
- wartość brutto

po wszystkich pozycjach podsumowanie zawiera sumę do zapłaty za całość.

Możliwe dalsze rozwinięcia:

1. Oprócz osobnych pozycji i jednej sumy na koniec wprowadź – jak na prawdziwych fakturach – osobne podsumowania dla poszczególnych stawek podatkowych. Zrób to wprowadzając osobną metodę `faktura.sumaDlaStawki(stawkaVAT)` i wypisz sumy dla poszczególnych stawek w `wydrukuj()`.
2. Zamiast `double` użyj klasy `BigDecimal`. Zaokrąglaj wszystkie wypisywane kwoty do dwóch miejsc po przecinku.