1. Podstawy Javy

Zadania opcjonalne dla tych, którzy chcą podciągnąć swoje kompetencje z podstaw programowania i języka Java.

Te zadania rozwiązujemy w formie programów z main, jak robiliśmy w Projekt2 w module czysta_java.

Zadanie 1.1 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 1.2 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 1.3 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 1.4 Funkcje na liczbach i napisach

Napisz poniższe funkcje **i sprawdź ich działania za pomocą testów jednostkowych JUnit**.

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 **fib**(int n)
   Zwraca n-tą liczbę Fibonacciego, gdzie liczby Fibonacciego są zdefiniowane następująco:
   fib(0) = 0
   fib(1) = 1
   fin(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 dziewięćdziesiątą liczbę Fibonacciego (jeszcze mieści się w long).
   Dla chętnych: wersja BigInteger – bez ograniczenia na wielkość wyniku.
3. boolean **palindrom**(String napis)
   Sprawdza czy napis jest palindromem (np. "kajak").
4. boolean **palindrom**(String napis, boolean normalizuj)
   W tej wersji, jeśli drugi parametr jest równy true, przed sprawdzeniem dokonywana jest „normalizacja” napisu, polegająca na tym, że napis jest zamieniany na małe litery i usuwane są z niego spacje. Wtedy np. "Kobyła ma mały bok" też jest palindromem. Gdy parametr jest równy false, sprawdzany jest oryginalny napis.

Zadanie 1.5 Klasy dla ogłoszeń *(przykład na dziedziczenie)*

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 **`Ogloszenie`** 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.

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

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

Możesz utworzyć więcej klas i różne definicje pomocnicze. Własne pomysły mile widziane.

Trzymaj się "standardowego podejścia Javy". W używanym IDE wygeneruj standardowe konstruktory i metody, dodaj inne metody według uznania. Napisz program, który tworzy i wypisuje kilka takich obiektów.

Podpowiedź – poszukajcie klas Osoba i Student w moich gotowych java_examples (pakiet p12 podstawy). Szczególną uwagę zwróćcie na konstruktory i zapis super(imie, nazwisko, wiek).

Zadanie 1.6 Hulajnoga

Stwórz klasę Hulajnoga, której obiekty reprezentują hulajnogi elektryczne o różnym zasięgu.

Wpisuję kod pokazujący, jak ma wyglądać użycie tej klasy. Można z tego zrobić program testujący (taki z main) albo zestaw testów jednostkowych JUnit (to polecam bardziej w kontekście kursu Android).

Dla ambitnych - po napisaniu tej klasy możesz dodać interfejs użytkownika w Android, który prezentuje bieżący stan baterii (stopień naładowania) i umożliwia wykonywanie operacji przejedz oraz naladuj.

```
// Tworzę hulajnogę o zasięgu 30 km
Hulajnoga h1 = new Hulajnoga(30);
System.out.println(h1);
int wynik;

// Etap 1 - rozładowywanie baterii gdy jeździ się hulajnogą
// Przejeżdżam hulajnogą 20 km
// Metoda ma zwracać (return) informację o ilości przejechanych kilometrów.
// Wynikiem powinno być 20, bo tyle udało się przejechać, a bateria powinna się "częściowo rozładować"
wynik = h1.przejedz(20);
System.out.println("przejdz(20), wynik = " + wynik); // 20

// Jeśli teraz ponownie spróbuję przejechać 20 km, to wynikiem będzie już tylko 10,
// bo hulajnodze udało się przejechać tylko 10 - tylko na tyle starczyło baterii.
wynik = h1.przejedz(20);
System.out.println("przejdz(20), wynik = " + wynik); // 10

// Jeśli teraz spróbuję przejechać cokolwiek, wynikiem powinno być 0
wynik = h1.przejedz(15);
System.out.println("przejdz(15), wynik = " + wynik); // 0

// Etap 2 - doładowywanie
// Metoda naladuj (bez parametrów!) powinna naładować baterię do pełna, czyli do tylu km, ile było podane na początku.
System.out.println("ładowanie h1");
h1.naladuj();

// Teraz przejedz powinno pozwolić na przejechanie 30 km
wynik = h1.przejedz(100);
System.out.println("przejdz(100), wynik = " + wynik); // 30

// Ale każdy obiekt hulajnoga może mieć podany inny zasięg:
Hulajnoga h2 = new Hulajnoga1(50);
```

2. Android – po pierwszych zajęciach

Propozycje prostych aplikacji Androida. Można zrealizować jako oddzielne projekty lub jako kolejne ekrany (aktywności) w ramach jednej aplikacji.

Zadanie 2.1 Kalkulator

Proponuję dwa pola typu EditText do wprowadzania liczb, cztery przyciski (lub więcej) dla poszczególnych operacji matematycznych, a poniżej pole typu TextView lub EditText na wyświetlenie wyniku. Można ograniczyć się do liczb całkowitych i pracować na typie long. Spróbuj obsłużyć dzielenie przez zero w taki sposób, że zamiast wyniku pojawi się tekst BŁĄD.

Zadanie 2.2 Przelicznik kosztów podróży

samochodem na podstawie ceny paliwa, spalania i dystansu. Dodatkowo możesz dodać pole na liczbę osób, na które dzielimy koszty podróży (domyślnie 1).

Poniżej zrzut ekranu takiej aplikacji, ale wykonanej w technologii Swing (na komputer).

Zadanie 2.3 Waluty

Uzupełnij przykład Waluty (lub utwórz „na czysto” nową wersję tego projektu – możesz wykorzystać moje klasy z pakietu waluty) o pobieranie archiwalnych kursów walut po wybraniu dnia z kalendarza.

Zadanie 2.4 Pogoda

Propozycja dodatkowego projektu po zakończeniu całego kursu, jeśli nie będzie to robione w czasie zajęć. Korzystając z możliwości serwisu open-meteo.com pobieraj aktualne informacje o pogodzie dla bieżącej lokalizacji (wykorzystaj lokalizację GPS).

Na następnej stronie wskazówki dot. tego projektu, które pisałem dla kursantów na „zwykłej Javie”, może się coś przyda...

Usługa open-meteo.com udostępnia dane pogodowe (stan bieżący oraz prognozę) w formacie JSON bez potrzeby rejestracji konta (czego wymaga większość takich serwisów).

Zapytanie pod adres https://geocoding-api.open-meteo.com/v1/search?name={miasto} zwraca listę znalezionych lokalizacji o podanej nazwie (lub w których nazwie jest podany fragment). Zwykle pierwsza lokalizacja jest tą najwłaściwszą. Dane lokalizacji obejmują m.in. długość i szerokość geograficzną.

Zapytanie pod adres https://api.open-meteo.com/v1/forecast?latitude={lat}&longitude={lon} zwraca dane pogodowe dla podanych współrzędnych geograficznych. Dane obejmują bieżącą pogodę oraz listę prognoz. Sami zbadajcie strukturę wynikowego JSONa - jest trochę bardziej skomplikowana, niż w przypadku walut.

Dodatkowo w zapytaniu można precyzować, czy chcemy dostać bieżącą pogodę, oraz jakie parametry pogodowe chcemy otrzymać w prognozie godzinnej. Przykładowe zapytanie dla przybliżonych współrzędnych Warszawy:
https://api.open-meteo.com/v1/forecast?latitude=52.23&longitude=23&current_weather=true&hourly=temperature_2m,relativehumidity_2m,windspeed_10m

Napisz program, który:

- pobiera od użytkownika nazwę miasta
- za pomocą pierwszego zapytania pobiera listę lokalizacji i wypisuje je ponumerowane, dla każdej z nich takie pola: name, country, population, longitude, latitude, elevation
- użytkownik wybiera jedną z nich (podaje numer pozycji na liście)
- program za pomocą drugiego zapytania pobiera dane pogodowe dla wybranej lokalizacji
- wypisuje kilka wybranych parametrów dla bieżącego stanu pogody
- wypisuje prognozę temperatury (pole temperature_2m) wraz z datami i godzinami, dla których są prognozowane.