Kurs Java Backend – ćwiczenia

1. Zadania w zakresie Java SE i „zwykłego programowania”

Można potraktować jako „wyzwania” i rozwiązywać sobie spokojnie podczas całego kursu...

Zadanie 1.1 Liczby tekstowo

Aplikacje użytkowe, zwłaszcza te związane z pieniędzmi, często muszą wypisywać liczbę lub kwotę w postaci tekstowej.

Stwórz „bibliotekę”, która będzie zajmować się tłumaczeniem liczb na postać tekstową.
W najprostszym podejściu może to być pojedyncza klasa posiadająca jedną lub kilka metod publicznych (mogą być statyczne). W miarę potrzeb można utworzyć więcej klas.

1. Podstawowa metoda powinna mieć postać:
   `public static String wartoscTekstowo(long liczba)`
   i powinna zwracać tekstową postać liczby w języku polskim.
2. Dodatkowo: także metodę, która w języku polskim prawidłowo odmienia słowo złotych, np. *dwadzieścia jeden złotych*, *dwadzieścia dwa złote*.
3. Dla chętnych: obsługa innych języków.
4. Jeśli znasz już temat testów, do metod napisz zestawy testów JUnit.

Zadanie 1.2 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”: 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.

Zgodnie z najnowszymi wzorcami ukryj konstruktor jako prywatny, a udostępnij zestaw metod statycznych, za pomocą których pobiera się obiekty (jak w LocalDate). Nie dopuszczaj do sytuacji, aby w mianowniku znalazło się zero – wyrzucaj wyjątek IllegalArgument. 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(int nom, int denom)
- static Fraction ofWhole(int i) – przyjmuje denom równe 1
- static final Fraction ZERO, ONE, HALF, ... – kilka stałych, podobnie jak w BigInteger
- int getNominator(), int getDenominator()
- String toString()
- equals + hashCode – w oparciu o wartości,

  do decyzji autora czy 3/4 i 6/8 mają być równe czy różne...
- int compareTo(Fraction other) – klasa powinna implementować interfejs Comparable
- Fraction shortened() – zwraca ułamek o tej samej wartości, maksymalnie skrócony (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 wartość typu double

Jako Number

Niech klasa Fraction będzie podklasą abstrakcyjnej klasy java.lang.Number, do której należą już Integer, Double, BigDecimal itd. Zaimplementuj wymagane metody.

Testy klasy Ułamek

Jeśli znasz temat testów jednostkowych, to w trakcie tworzenia wyżej opisanej klasy twórz też testy jednostkowe do poszczególnych operacji.

Operacje na strumieniach

Utwórz przykładową tablicę lub listę ułamków i za pomocą operacji na strumieniach (filter, map, reduce) wykonaj kilka przykładowych operacji. Moje propozycje (każdy punkt to nowa operacja startująca od listy ułamków):

- wypisać tylko ułamki właściwe (licznik < mianownik),
- zamienić na strumień / listę doubli i obliczyć średnią jako double,
- obliczyć precyzyjnie (jako ułamek) sumę i średnią elementów,
- znaleźć największy i najmniejszy element,
- posortować wg wartości,
- posortować wg mianowników,
- pogrupować ułamki po mianownikach,
- skrócić a następnie pogrupować ułamki po mianownikach i wypisać te grupy w postaci posortowanej malejąco wg mianowników, a w obrębie każdej grupy rosnąco wg liczników.

Zadanie 1.3 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 pomóc 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 slowo = sc.next();
    ...
}
```

Rozwiązania można napisać w oparciu o pętle 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 1.4 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.

2. Wątki

Zadanie 2.1 Gra na refleks

Program w jednym wątku losuje liczbę całkowitą A od 1 do 10 włącznie, wypisuje "A wynosi …", czeka losową ilość czasu od 0 do 2 sekund, ponownie losuje liczbę A, wypisuje, czeka, i tak w kółko. Niezależnie (w osobnym wątku) program losuje liczbę całkowitą B od 1 do 10 włącznie, wypisuje "B wynosi …", czeka losową ilość czasu od 0 do 2 sekund, ponownie losuje liczbę B, wypisuje, czeka i tak w kółko. Czekanie i losowanie jest niezależne dla liczb A i B. Użytkownik w dowolnym momencie może wcisnąć enter. Jeśli w tym momencie A i B są równe, to program wypisuje "trafiony" i kończy działanie. Jeśli A jest inne niż B to wypisuje "pudło" i gra toczy się dalej.

Zadanie 2.2 Suma tablicy jedno / wielowątkowo

Napisz program, który tworzy dużą tablicę liczb int (dobierz rozmiar, który mieści się w pamięci), wypełnia ją losowymi wartościami (użyj klasy Random lub ew. ThreadLocalRandom, to może chwilę potrwać). Następnie oblicz sumę wszystkich elementów tablicy **jako BigInteger**.

Niech program najpierw oblicza sumę w tradycyjny sposób i mierzy czas wykonania (wykorzystaj System.currentTimeMillis), a następnie niech obliczy sumę używając kilku wątków i również zmierz czas tego działania (każdy wątek zajmuje się sumowaniem osobnego fragmentu tablicy, a następnie main zbiera częściowe wyniki). Porównaj czas wykonania. Na wieloprocesorowym/wielordzeniowym sprzęcie wersja z wątkami powinna działać szybciej, ale dawać równie poprawny wynik...

3. Serwlety

Zadanie 3.1 Czat w serwletach

Za pomocą serwletów i JSP zaimplementuj prosty czat:

- użytkownik na początku wybiera swój *nick*,
- następnie może w formularzu wpisywać komunikaty, które będą widoczne dla wszystkich,
- przy każdym komunikacie pojawia się *nick* autora, wszyscy widzą komunikaty wysłane przez wszystkich.

Komunikaty wysyłane przez innych stają się widoczne dopiero po wysłaniu własnego komunikatu lub odświeżeniu strony – tak to ma działać.

> Wyświetlanie komunikatów natychmiast wymagałoby użycia JavaScript na stronie w połączeniu z technologią websocket na serwerze lub ewentualnie niewydajnej strategii odpytywania/odświeżania.

Dane mogą być przechowywane w pamięci, tylko na czas działania aplikacji. Trzeba użyć odpowiednich zasięgów dla zmiennych przechowujących listę komunikatów oraz nick bieżącego użytkownika. Nick ma być pamiętany w sesji. Pamiętaj też o tym, że serwlety działają wielowątkowo.

Do Twojej decyzji zostawiam, czy użyjesz samych serwletów (z printowaniem HTML-a w Javie), czy stron JSP.