Zaawansowane programowanie obiektowe
Lab. 4
(Collator; JSON; formatowanie liczb)

1. (1 pkt)
   Napisz klasę zawierającą metody sortujące napisy z uwzględnieniem alfabetu polskiego (np. „Łukasz" ma być między „Lucyna" a „Marek").
   Wskazówka: wykorzystaj klasę java.text.Collator.

   Konkretnie napisz 3 metody sortujące:

   public static void sortStrings(Collator collator, String[] words)
   – sortującą napisy ręcznie i naiwnie, z użyciem sortowania przez wstawianie (*insertion sort*),

   public static void fastSortStrings(Collator collator, String[] words)
   i
   public static void fastSortStrings2(Collator collator, String[] words)
   – sortujące napisy z użyciem Arrays.sort(…).
   Różnica między tymi dwiema metodami jest taka, że fastSortStrings ma używać anonimowego obiektu komparatora, zaś fastSortStrings2 ma wykorzystać funkcję lambda.

   W testach (z użyciem JUnit) porównaj zgodność wyników zwracanych przez wszystkie te 3 funkcje, a także wyświetl wyniki na konsoli dla następującej tablicy:
   ```
   String[] names = {"Łukasz", "Ścibor", "Stefania", "Darek", "Agnieszka",
                     "Zyta", "Órszula", "Świętopełk"};
   ```

   Wykonaj również test wydajnościowy tych 3 metod, sortując powyższą tablicę imion w pętli 100 tys. razy (oczywiście na starcie ma być za każdym razem nieposortowana). Tym razem nie wypisuj tablicy na ekranie. Wykorzystaj metodę System.nanoTime().

2. (1,5 pkt)
   Zaimplementuj test ze znajomości słówek angielskich. Baza ma liczyć 10 pytań, z których do testu losujemy bez powtórzeń 5. „Pytaniem" ma być słowo polskie, odpowiedzią – wpisywane z tzw. palca, ale w okienku dialogowym (z użyciem JavaFX lub Swing), słowo angielskie. Na końcu testu należy podać wynik (tj. ile pytań poprawnych) oraz zużyty czas, z dokładnością do 0,01s.

   Jednemu słowu polskiemu może odpowiadać kilka (dozwolonych) tłumaczeń, np. krzyczeć -> shout / cry / scream. Program powinien być niewrażliwy na małe / wielkie litery (czyli cry / CRY / Cry etc. są równoważne).

   Co więcej, dopuszczamy 1 błąd Levenshteina (przypomnij sobie zadanie nr 1 z lab02), ale używamy standardowej (a nie ważonej, jak w tamtym zadaniu) miary Levenshteina. Jeśli odpowiedź jest z jednym błędem, to dajemy za nią 0,5 pkt. Jeśli jest bezbłędna, to 1 pkt.

Przykłady:

krzyczeć --> cry (1 pkt.)
krzyczeć --> SoUT (0,5 pkt) (gdyż jedna z prawidłowych odpowiedzi to „shout”; małe/duże litery nie mają znaczenia)
krzyczeć --> cri (0,5 pkt)
krzyczeć --> cray (0,5 pkt)
krzyczeć --> crie (0 pkt) (gdyż tu są już 2 błędy w sensie miary Levenshteina)

„Baza” pytań i odpowiedzi, w formacie JSON, zawarta jest w pliku PolEngTest.json (należy go najpierw „ręcznie” utworzyć! Postać niech będzie czytelna, więc nie jeden bardzo długi wiersz tekstu), który należy odczytać (zdeserializować) przy użyciu biblioteki google-gson: https://github.com/google/gson (lub innej użytecznej biblioteki).
Dokumentacja Gson:
http://www.javadoc.io/doc/com.google.code.gson/gson/2.8.6
Format JSON jest przedstawiony np. pod
http://www.tutorialspoint.com/json/json_tutorial.pdf .

Przebieg egzaminu (tj. zbiór par napisów: pytanie-odpowiedź) ma być również zapisany w pliku JSON o nazwie imie_nazwisko.json, w analogicznym formacie jak plik wejściowy.

Napisz testy JUnit (nie tylko dla sprawdzania odl. Levenshteina).

3. (1 pkt)
Proszę napisać funkcję o nagłówku:
List<String> formattedNumbers(List<Double> nums, int group, char separator, int nDigits, boolean padding)

która dla wejścia
List<Double> li = List.of((double)-5100, 43.257, (double)200000, 2000000.5);
List<String> fn = formattedNumbers(li, 2, ',', 2, true);

spełni następujące testy JUnit:

```
assertTrue(fn.size() == 4);
assertTrue(fn.get(0).equals("    -51,00.00"));
assertTrue(fn.get(1).equals("        43.26"));
assertTrue(fn.get(2).equals("  20,00,00.00"));
assertTrue(fn.get(3).equals("2,00,00,00.50"));
```

Zmieńmy parametry (ale li bez zmian):

```
List<String> fn2 = formattedNumbers(li, 3, '|', 2,
false);
assertTrue(fn2.size() == 4);
assertTrue(fn2.get(0).equals("   -5|100"));
assertTrue(fn2.get(1).equals("       43.26"));
assertTrue(fn2.get(2).equals("  200|000"));
```

```
assertTrue(fn2.get(3).equals("2|000|000.5"));
```

Wskazówka: sprawdzenie czy liczba x typu double jest de facto liczbą całkowitą: x == Math.rint(x) (przeczytaj opis metody rint w dokumentacji).