Zaawansowane programowanie obiektowe
Lab. 2

1. (1.5 pkt) Zaimplementuj funkcję
   double LevQWERTY(String s1, String s2),
   która zwraca ważoną odległość Levenshteina między napisami s1 i s2, gdzie wagi zależne są od wzajemnego położenia pary znaków na klawiaturze.
   Konkretniej, odl. Levenshteina bazuje na 3 elementarnych operacjach: wstawienia znaku (ang. *insertion*), usunięcia znaku (ang. *deletion*) oraz zastąpienia znaku innym (ang. *substitution*). W naszym przypadku waga operacji insercji i delecji ma wynosić 1, natomiast waga substytucji wynosi:
   - 0.5, jeśli odnośna para znaków sąsiaduje w rzędzie na klawiaturze,
   - 1, w przeciwnym przypadku.

   Zakładamy, że s1 i s2 mogą zawierać tylko małe litery łacińskie oraz spacje.

   Przykłady:
   LevQWERTY("kot", "kita") == 1.5 (1 insercja (a) + 1 substytucja znaków sąsiadujących w rzędzie (o <--> i)).
   LevQWERTY("drab", "dal") == 2 (1 delecja (r) + 1 substytucja znaków niesąsiadujących w rzędzie (b <--> l)).

   Napisz testy jednoskowe z użyciem jUnit sprawdzające poprawność napisanej funkcji.

   Wskazówka: zastosuj tablicę asocjacyjną z małymi literami łacińskimi jako kluczami oraz zbiorami liter z nimi sąsiadujących jako wartościami.

   Formuła programowania dynamicznego dla obliczania odl. Levenshteina + przykład: http://szgrabowski.kis.p.lodz.pl/Alg15/lecture09.ppt Slajdy 33–36.

2. (3 pkt) Zasymuluj wyścig kolarski na czas. W wyścigu uczestniczy 15 kolarzy. Co 1 minutę (czasu symulowanego) startuje następny kolarz; jego czas, wyrażony w całkowitej liczbie sekund, jest liczbą losową o rozkładzie Gaussa ze średnią 300s i odch. standardowym 30 (ale czas nie może być krótszy niż 250s ani dłuższy niż 370s, tj. jeśli wylosuje się np. 246s, to przyjmujemy 250s, a jeśli wylosuje się 381s, to przyjmujemy 370s).

   Nazwiska kolarzy (bez imion) mają być wzięte losowo z pliku: http://szgrabowski.kis.p.lodz.pl/zpo17/nazwiska.txt (który jest odczytywany spod swojego URLa, tj. proszę go nie ściągać wcześniej na dysk), nie dopuszczamy do powtórzeń nazwisk. Plik zakodowany jest w UTF-8.

   W symulacji 1 sekunda czasu rzeczywistego ma odpowiadać 25s czasu symulowanego.
   Na bieżąco ma być pokazywany czas zawodnika, który właśnie ukończył wyścig oraz aktualna posortowana czołówka 3 zawodników (ich nazwiska i czasy). Na samym początku ta czołówka będzie mniej liczna (0..2 kolarzy).

   Dodatkowo każde zdarzenie symulacji ma być logowane do pliku (java.util.logging.Logger, java.util.logging.LogManager).

Wskazówki:
1) do losowania nazwisk bez powtórzeń możesz wykorzystać zbiór (java.util.Set),
2) czytanie bezpośrednio spod URLa:
http://docs.oracle.com/javase/tutorial/networking/urls/readingURL.html ,
3) uruchamianie „zadań” w regularnych odstępach czasowych: java.util.Timer bądź współczesną implementację:
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html
3) do znajdowania aktualnej czołówki wykorzystaj np. kolejkę priorytetową (java.util.PriorityQueue):
https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html
4) Generowanie liczb losowych – rozkład Gaussa
https://www.javamex.com/tutorials/random_numbers/gaussian_distribution_2.shtml