Zadanie domowe nr 1

Świat oglądany przez okulary miłośnika komunikacji miejskiej[1]

Programowanie Obiektowe, Informatyka

Wersja 1.02

Ogólny opis

Zaprojektuj w Javie zestaw klas służących do (bardzo uproszczonej) symulacji ruchu miejskiego. Zadbaj o obiektowość swojego rozwiązania. W szczególności o możliwość rozszerzania symulacji o nowe elementy.

W symulacji mają być uwzględnione m.in. następujące elementy:

- pasażerowie,
- pojazdy (w tej symulacji tylko tramwaje),
- linie tramwajowe,
- przystanki .

Pojazdy mogą być różne, ale każdy ma numer boczny i swoją linię. Linie mają swoje trasy opisane w danych (p. dane). W trasie jest opisany również czas postoju na pętli. Jeden pojazd zawsze jeździ tą samą trasą. Pojazd przejeżdża swoją trasę w jedną stronę, zatrzymuje się na pętli, potem wraca (tą samą trasą) i znów ma postój na pętli. Tramwaje są pojazdami. Pierwsze tramwaje wyjeżdżają na trasę o 6. Po 23 już tylko kończą rozpoczęte przejazdy (co w przypadku pechowego motorniczego, który wyjechał z pętli o 23:00, oznacza konieczność zrobienia pełnego kółka, tramwaje nie zostają na noc na pętli na drugim końcu trasy). Tramwaje jednej linii są wypuszczane rano na trasę z obu jej stron: połowa z jednego końca, połowa z drugiego (jeśli ich liczba jest nieparzysta, to o jeden więcej z pierwszego końca). Tramwaje wyjeżdżają rano na trasę w równych odstępach, obliczanych jako czas przejazdu w obie strony z postojami na pętli, dzielony przez liczbę tramwajów na tej trasie). Każdy tramwaj ma swoją pojemność, jednakową dla wszystkich tramwajów (parametr zadania).

Trasa linii tramwajowej składa się z przystanków. Przystanki mają swoje nazwy oraz zadaną (wspólną dla wszystkich przystanków) pojemność. Nie rozróżniamy tu przystanków w jedną i drugą stronę.

Każdy *pasażer* mieszka w pobliżu jakiegoś przystanku. Codziennie rano, o losowej godzinie pomiędzy 6 a 12 (godzina jest losowana każdego dnia na nowo) idzie na swój przystanek. Jeśli na przystanku nie ma już miejsc, to rezygnuje z podróżowania tego dnia. Wpp. czeka na przystanku na tramwaj (dowolnej linii, w dowolną stronę). Po przybyciu tramwaju próbuje do niego wsiąść, jeśli to się uda, to wybiera losowo jeden z przystanków pozostałych na trasie (w jednym kierunku), tam jedzie i próbuje wysiąść. Jeśli to się nie uda, to kontynuuje jazdę, z nadzieją, że za którymś nawrotem w końcu się uda. Gdy już znajdzie się na przystanku zaczyna tę samą procedurę: czeka na tramwaj i próbuje do niego wsiąść. Jeśli się nie uda, to czeka na nastęny. (W trosce o dobrostan pasażerów zakładamy, że biorą z domu termos i zapas kanapek

[1] Inspirowane "Słówkami" Tadeusza Boya-Żeleńskiego.

lub kupują coś niezdrowego z automatu na przystanku, czekając na tramwaj - jednak w implementacji tego zadania pomijamy wszystkie te kwestie).

Gdy tramwaj przyjeżdża na przystanek, najpierw wypuszcza pasażerów, którzy chcieli tu wysiąść (części może się to nie udać, gdy na przystanku nie ma już wolnych miejsc). Kolejność wypuszczania pasażerów jest dowolna. Następnie do tramwaju próbują wsiąść oczekujący pasażerowie (znów, części z nich może się to nie udać, gdy w tramwaju nie będzie już miejsc).

Symulacja trwa zadaną (p. dane) liczbę dni. Zaczyna się od poniedziałku (ale w tej wersji zadania nie ma to znaczenia).

Gdy kończy się dzień symulacji, wszyscy pasażerowie wracają do swoich domów (np. pieszo lub taksówką, tego już nie symulujemy, czyli nie mamy pojazdu taksówka w symulacji, a rano wszyscy są w domu). Tramwaje nocą dojeżdżają na właściwe końce swoich tras (nie symulujemy już tego, po prostu rano są tam gdzie być powinny).

Dodatkowe założenia:

- Pojazdy nie kursują po północy (zakładamy, że przejazd trasy trwa krócej niż godzinę, a po 23 pojazdy już nie ruszają w trasę z pasażerami, mogą co najwyżej wrócić do zajezdni).

Wynik programu

Program powinien wypisać trzy rzeczy:

1. Wczytane parametry.
2. Szczegółowy zapis symulacji, po jednym wierszu na każde zdarzenie. Każdy wiersz powinien zawierać na początku czas zdarzenia (dzień symulacji i godzinę) i dwukropek potem opis zdarzenia (np. 46, 15:23: Pasażer 213 wsiadł do tramwaju linii 1 (nr bocz. 14) z zamiarem dojechania do przystanku Centrum.).
3. Statystyki symulacji. Co najmniej:
   - łączną liczbę przejazdów pasażerów (każdy przejazd jednego pasażera jednym pojazdem liczymy osobno),
   - średni czas czekania na przystanku (ile średnio czekał ktoś, kto był na przystanku),
   - dla każdego dnia symulacji:
     - łączną liczbę przejazdów (liczonych j.w.),
     - łączny czas czekania na przystankach (suma po wszystkich czekających osobach).

Wyniki należy wypisać na standardowe wyjście (za pomocą `System.out`).

Organizacja symulacji

Symulację przeprowadza się za pomocą kolejki zdarzeń. To z punktu widzenia programu linia czasu --- lista zdarzeń posortowana niemalejąco zwn. czas ich wystąpienia. Każdy obiekt wiedząc, że ma coś zrobić w przyszłości, wrzuca do kolejki stosowne zdarzenie ze sobą i z czasem, kiedy zdarzenie ma

nastąpić. Symulacja polega na pobieraniu z kolejki pierwszego (tj. o najniższym czasie wystąpienia) zdarzenia i proszeniu pobranego w nim obiektu o zareagowanie. Dzieje się tak przez zadany czas symulacji (nie: czas rzeczywisty) lub aż kolejka stanie się pusta. W naszej symulacji, ze względu na to, że w nocy nic nie jeździ, w końcu kolejka zdarzeń każdego dnia powinna się opróżnić.

Czas w naszej symulacji mierzymy w wirtualnych (nie: rzeczywistych) minutach liczonych od początku dnia lub początku symulacji (jak komu wygodniej, akurat w tym zadaniu nie ma zdarzeń przechodzących z dnia na dzień). Zakładamy ziemską, zaokrągloną długość doby (60*24 minut).

Kolejkę zdarzeń można zaimplementować na różne sposoby. Możliwe realizacje to np.:

- Posortowana tablica, pamiętająca do którego miejsca jest wypełniona. Jej rozmiar powinien być podwajany w sytuacji, gdy zabraknie w niej miejsca na kolejne zdarzenia.
- Posortowana lista zdarzeń (lista w znaczeniu takim jak w pierwszym semestrze na C, tj. zestaw elementów, z których każdy pamięta swojego następnika).
- Sterta (ang. heap) znana np. z ASD.

W tym zadaniu oczekujemy dowolnej z tych implementacji (wszystkie będą tak samo punktowane), dostarczających operacji wstawienia nowego zdarzenia do kolejki, pobrania pierwszego i sprawdzenia, czy kolejka jest pusta. Przy wstawianiu kilku zdarzeń o tym samym czasie zdarzenia później wstawione powinny być później wydawane z kolejki.

Wymagamy też, by w programie był interfejs dla kolejki zdarzeń (oczywiście implementowany przez Państwa kolejkę), tak by można było łatwo podmienić jedną implementację na inną (w tym zadaniu wymagamy tylko jednej implementacji).

Próba pobrania zdarzenia z pustej kolejki powinna być wychwytywana asercją.

Zdarzenia mogą tworzyć hierarchię.

Oczywiście w bibliotece standardowej Javy są dostępne klasy realizujące kolejkę zdarzeń, ale w tym zadaniu wymagamy Państwa własnej implementacji (jednej z podanych).

Opis danych

Dane dla symulacji należy wczytać ze standardowego wejścia (p. dodatkowe wymagania).

Postać danych (teksty w nawiasach kątowych opisują kolejne dane, zaś teksty w nawiasach okrągłych ich typy):

```
<Liczba dni symulacji (int)>
<Pojemność przystanku (int)>
<Liczba przystanków (int)>
Teraz dla każdego przystanku
   <Nazwa przystanku (String)>
<Liczba pasażerów (int)>
```

```
<Pojemność tramwaju (int)>
<Liczba linii tramwajowych (int)>
Teraz dla każdej linii (linie są numerowane kolejnymi liczbami naturalnymi):
   <Liczba tramwajów tej linii, długość trasy, (int, int)>
   potem trasa w postaci par:
      <nazwa przystanku, czas dojazdu (string, int)>.
```

Ostatni czas w opisie linii to czas postoju na pętli, zakładamy, że taki sam dla obu pętli. Na przykład:

[P1, P2, P3], [1, 2, 3]

oznacza czasy przejazdu P1<->P2 1 min, P2<->P3 2 min oraz postój na pętlach P1 i P3 po 3 minuty.

Uwagi:

- Zakładamy, że wszystkie napisy są ciągami znaków różnych od białych spacji (czyli nazwa miejsca *MIMUW* oraz MIM-UW są dozwolone, ale *MIM UW* już nie).
- Przedstawiona postać danych zakłada podział na wiersze --- wtedy dane dużo łatwiej nam (istotom białkowym) tworzyć je lub edytować. Państwa program nie musi jednak analizować/egzekwować podziału na wiersze, czyli wystarczy po prostu czytać po kolei kolejne liczby lub napisy, nie sprawdzając ich położenia w wierszu. Jeśli jednak ktoś chce wykorzystać w swoim programie zasugerowany w treści zadania podział na wiersze, to może --- dane testowe będą (tak jak w przykładzie dalej) podzielone na wiersze.

Przykładowe (minimalne) dane:

```
0
0
0
0
0
0
```

Przykładowe (nieco większe) dane:

```
3
10
3
Banacha
Krakowskie
Centrum
15
5
1
2 3 Banacha 3 Centrum 2 Krakowskie 10
```

Dodatkowe wymagania

Do **wszystkich** losowań w symulacji należy stosować **tylko** klasę `Losowanie`, mającą zdefiniowane następujące metody:

`public static int losuj(int dolna, int gorna)` - daje losową liczbę całkowitą z obustronnie domkniętego przedziału dolna..górna.

Klasa ta powinna znaleźć się w tym samym pakiecie, co główna klasa programu (ta z `main`).

Dane do symulacji należy czytać za pomocą klasy `java.util.Scanner`. Przykład użycia tej klasy prezentuje poniższa metoda. Czyta ona ze standardowego wejścia liczbę słów, a potem same słowa. Wczytane słowa wypisuje na wyjściu po zakończeniu czytania.

```
public static void przyklad() {
    Scanner sc = new Scanner(System.in);
    int ile = sc.nextInt();
    String[] napisy = new String[ile];
    for (int k = 0; k < napisy.length; ++k)
        napisy[k] = sc.next();
    for (int k = 0; k < napisy.length; ++k)
        System.out.println(napisy[k]);
}
```

Dodatkowe uwagi

- Prosimy pamiętać, że to zadanie z programowania obiektowego, zdecydowanie nie należy się obawiać korzystania z klas, obiektów, konstruktorów, zakresów widoczności itp. Np. idealnie działający program z jedną tylko klasą będzie odczytany jako nieograniczona sympatia dla okrągłych liczb parzystych mających nieskończenie wiele dzielników.
- To większe zadanie, prosimy o rozwiązywanie go w sposób przyrostowy. Np. dojście do wersji czytającej pierwsze powyższe dane jest dobrym etapem tworzenia rozwiązania. Przejście do czytania drugich danych z tego zadania może być jednym z kolejnych kroków (niekoniecznie następnym). Uruchomienie symulacji z samymi tramwajami (jeszcze bez pasażerów), także wydaje się rozsądnym etapem pracy.
- Zadanie zostało tak zaprojektowane (dokładniej: okrojone :)), żeby dało się je łatwo zrobić, korzystając z tablic. Ponieważ nie znamy jeszcze kontenerów ze standardowej biblioteki Javy, nie możemy ich w tym zadaniu wymagać. A ponieważ wszyscy powinni mieć takie samo zadanie do zrobienia, to **nie wolno** w tym zadaniu korzystać z kontenerów innych niż tablice.
- Możliwe implementacje kolejki priorytetowej zostały wskazane wcześniej, gdyby było potrzebne szybkie wyszukiwanie można zastosować (znane nam już z klasówki) sortowanie tablic (lub można zapisać własne) oraz wyszukiwanie binarne.
- Przyjmujemy numerowanie wszystkiego (linie, nry boczne, ...) od zera.

Życzymy powodzenia

Historia zmian:

- 1.0: 26 IV 2024, pierwsza wersja.
- 1.01: 27 IV 2024: korekty literówek.
- 1.02: 27 IV 2024: dodany podtytuł, drobne korekty i dodatkowe wyjaśnienia (przykład wyjaśniający znaczenie elementów opisu linii, założenie o numerowaniu od 0).