Text Stream API

Co raz większą popularność zdobywają API oparte o pomysł strumienia. Dane są tu traktowane jak strumień (potencjalnie nieskończony) pojawiających się obiektów, które są w zdefiniowany przez użytkownika sposób transformowane by ostatecznie dać pożądany wynik. Samo pojęcie strumienia jest dość ogólne. Istnieją różne podejścia do takich API (np. rozróżnienie między Pull - sami wyciągamy dane ze strumienia kiedy chcemy a Push - kiedy pojawia się nowy obiekt to zostajemy o tym poinformowani i reagujemy na to jak na zdarzenie). Przykładami takich API są LINQ w C#, Java 8 Stream API a także bardzo rozbudowany zbiór bibliotek Reactive Extensions (dla różnych języków, np. RxJava, RxJS itd.), który korzysta z wielu innych pomysłów (oraz wspomnianego podejścia Push) ułatwiających między innymi programowanie asynchroniczne.

Podstawowe pomysły na architekturę w tych API są dość proste, jednak zbyt skomplikowane na jedno laboratorium. Dlatego też postaramy się zaimplementować małą cegiełkę tych monumentów. Naszym zadaniem będzie zaimplementowanie strumienia typu Pull (czyli będziemy z niego wyjmować wartości). Dodatkowo, dla uproszczenia, nasze metody będą transformowały tę wartości w sposób szczególny (dokładniej: będą operowały na charach i zakładały że zawsze otrzymają chara). Wspomniane wcześniej API zazwyczaj zawierają szereg bardzo ogólnych metod transformujących (np. Map, Filter czy Reduce).'

Projekt zawiera:

1. Definicję interfejsu IStream, który zawiera metody:
   - Character ReadNext() - wczytaj kolejny znak ze strumienia,
   - boolean AtEnd() - mówi, czy wczytaliśmy cały strumień
2. Implementację tego interfejsu w postaci klasy StringStream.
3. Metodę Main, która tworzy instancję IStream (oraz drugą, wykorzystaną w późniejszym zadaniu) i wypisuje to, co zwróci ReadNext().

Zadanie polega na tym, żeby zaimplementować swoje operacje na strumieniu, które pozwolą nam na transformację wejściowego ciągu znaków w sposób sekwencyjny i otrzymaniu na końcu również ciągu znaków.

1. Dodaj możliwość transformacji wartości emitowanej przez strumień w taki sposób, żeby wszystkie wielkie litery zostały zamienione na małe. W funkcji Main wywołaj tę transformację tak, żebyśmy mogli zobaczyć wynik.

2. Stwórz tym razem transformację zamieniającą litery małe na wielkie. Tak jak ostatnio, wywołaj tę transformację (może być zamiast poprzedniej lub po poprzedniej - tak żebyśmy zauważyli wynik).

3. Dodaj transformację zamieniającą wszystkie spacje w stringu na znaki podkreślenia ('_'). Ponownie wywołaj tę transformację.

4. Dodaj transformację, która łączy dwa strumienie (jako drugi wykorzystaj zadeklarowany już w metodzie Main strumień "another"). Łączenie polega na tym, że najpierw jest wczytywany pierwszy strumień a gdy ten się skończy to dopiero drugi.

5. Stwórz nowe strumienie które będą takie jak pierwotne strumienie „stream" i „another". Dokonaj takiego szeregu transformacji, aby strumień wynikowy był połączeniem strumienia „stream" pisanego samymi małymi literami ze strumieniem „another" pisanym samymi wielkimi literami.

6. Dodaj możliwość usuwania samogłosek (tj. po tej operacji w wynikowym strumieniu będą tylko spółgłoski).

7. Dodaj możliwość zamiany pierwszej litery po kropce na wielką.

8. Dodaj szyfrator i deszyfrator dla szyfru Cezara. Szyfr Cezara polega na tym, że przesuwamy wszystkie litery zadanego ciągu znaków o ustaloną liczbę liter w alfabecie. Przykład: "mini" dla liczby szyfrującej 3 da nam "plql".

9. Dodaj operację która w wyniku da na zmianę wielką i małą literę.

10. Wczytaj tylko N pierwszych znaków.

11. Pomiń pierwsze N znaków.