Wzorce Projektowe, lab. 9

Temat: Praktyczne zastosowanie wzorca **Flyweight**.

Twoje zadanie to implementacja programu do przetwarzania napisów dostarczonych przez konsolę lub z pliku.

1. Zacznij od utworzenia klasy **Znak**. Ma ona przechowywać w sobie dwa prywatne pola: **znak** typu char oraz **czyLitera** typu boolean. Konstruktor **Znaku** ma przyjmować jeden argument typu char i go przetwarzać:
   - jeśli jest to duża litera, to trzeba zamienić ją na małą, zapisać do pola **znak** oraz ustawić pole **czyLitera** na *true*,
   - jeśli jest to mała litera, to trzeba przypisać ją do pola **znak** oraz ustawić pole **czyLitera** na *true*,
   - jeśli jest to znak niebędący literą, to należy przypisać go do pola **znak** oraz ustawić pole **czyLitera** na *false*.

   W realizacji tego zadania pomocna może okazać się tablica znaków ASCII, którą możesz znaleźć w internecie i przejrzeć kody poszczególnych liter (załóżmy, że tekst nie będzie zawierał polskich znaków). Ostatecznie nasze **Znaki** będą albo małymi literami, albo znakami nieliterowymi – informacja o wielkości litery nie jest nam potrzebna. Oprócz tego w klasie powinna znaleźć się metoda `void wypisz(boolean czyDuzy)` która wypisze dużą reprezentację litery przy spełnieniu warunków **czyDuzy** oraz **czyLitera** lub przechowywaną w polu **znak** literę małą albo inny znak nieliterowy. Ostatnia metoda w naszej klasie to `boolean porownaj(char c)`, która zwróci *true* jeśli mamy w obiekcie dokładnie ten sam znak lub jego małą reprezentację.
2. Dodaj klasę **FlyweightFactory** z prywatną **HashMapą** naszych znaków (można posłużyć się slajdem 10 z pliku flyweight.pdf). Ma ona mieć bezargumentowy konstruktor inicjalizujący **HashMapę** oraz metodę `Znak getZnak(char z)` do zwracania odpowiednich **Znaków**. UWAGA! Zauważ, że mamy jedną reprezentację klasy **Znak** dla dużej i małej litery – obsłuż zatem ten przypadek, aby nie tworzyć jej nadmiarowych instancji. Wersja z wykładu doprowadziłaby do umieszczenia w mapie zarówno pary *<'a', new Znak('a')>*, jak i *<'A', new Znak('A')>*.
3. Utwórz klasę **BlokTekstu**, która będzie zawierała w sobie **ArrayListę Znaków** oraz reprezentację klasy **FlyweightFactory** z punktu 3. Jej fundament to 2 metody `void add(String s)` oraz `void add(char c)`, które będą dodawały do wewnętrznej **ArrayListy** obiekty klasy **Znak** zwrócone przez nasz **FlyweightFactory. BlokTekstu** ma mieć też trzy konstruktory: `BlokTekstu()` inicjalizujący dwa wspomniane wyżej pola, `BlokTekstu(String text)` , który odwoła się do pierwszego konstruktora, a następnie doda do **ArrayListy** dostarczony napis używając do tego odpowiedniej metody, a także `BlokTekstu(File f)` do obsługi wejścia z pliku. Ostatni konstruktor ma odczytywać z pliku cały tekst i dodawać do **ArrayListy**. Pamiętaj o obsłudze wyjątków!
4. Dodaj do klasy **BlokTekstu** trzy metody: `void wypiszDuze(); void wypiszMale(); void wypisz();`. Pierwsza ma wypisywać na konsoli przechowywany tekst w postaci samych dużych liter, druga w postaci samych małych liter, a trzecia zgodnie z zasadą: duża litera na początku zdania, a reszta liter mała. Kolejne znaki mają być rozpisywane obok siebie uwzględniając przejścia do nowych linii – jeśli w pliku znajdzie się wiersz, to Twój program będzie wiedział, kiedy kończą się poszczególne wersy.

5. W funkcji *main()* znajdującej się w głównej klasie **Flyweight** utwórz obiekt klasy **BlokTekstu**, poinformuj użytkownika, że czekasz na dostarczenie napisu w konsoli i wczytaj go. Wywołaj kolejno wszystkie trzy metody do wypisywania, a następnie poinformuj użytkownika, że oczekujesz ścieżki do pliku. Przetwórz ten plik i ponownie wywołaj wszystkie metody do wypisywania.

UWAGI:

- Wszystkie klasy i interfejsy muszą być w oddzielnych plikach.
- Wszystkie pola mają być prywatne, a wszystkie metody publiczne.
- Robiąc kilka konstruktorów staraj się nie kopiować kodu – pamiętaj, że możesz jednym konstruktorem zawołać inny, a potem obudować dodatkowym kodem.