21 marca 2018

Programowanie obiektowe

Lista 5.

Poniższe zadania mają być zaimplementowane w Javie. Dla każdego zadań proszę podać krótki program ilustrujący możliwości zaimplementowanych klas.

Zadanie 1. Zaimplementuj kolekcję przechowującą elementy w kolejności *rosnącej* wraz z metodami (lub właściwościami) dodania elementu, pobrania elementu (z jego usunięciem) oraz wypisania wszystkich elementów. Przyjmij, że przy pobieraniu elementu pobierany jest zawsze najmniejszy. Załóż, że elementy przechowywane w tej kolekcji muszą implementować interfejs umożliwiający porównywanie elementów (może to być standardowy interfejs `Comparable<T>`).

Zaimplementuj również dowolną hierarchię klas implementującą interfejs `Comparable<T>` (lub inny zaproponowaną przez Ciebie), zawierającą przynajmniej cztery klasy. Może to być np. hierarchia klas reprezentująca stopnie wojskowe bądź klasy reprezentujące figury geometryczne (interpretacja porównania: figura f_1 jest mniejsza od figury f_2 jeśli f_1 jest wewnątrz f_2).

Zwróć uwagę, aby implementacja tej hierarchii klas przestrzegała omówionej na wykładzie zasady otwarte–zamknięte, tj. aby można było dodać klasę (np. reprezentującą nowy stopień wojskowy) implementującą `Comparable<T>` bez konieczności zmiany implementacji w pozostałych klasach.

Zadanie 2. Wyrażenia arytmetyczne można reprezentować jako drzewa, gdzie w liściach pamiętane są liczby, a w węzłach symbole operacji arytmetycznych. Zaimplementuj w Javie odpowiednie klasy reprezentujące węzły i liście takiego drzewa jako podklasy klasy ***Wyrażenie***. W każdej klasie zdefiniuj metodę

```
public int oblicz();
```

obliczającą wartość wyrażenia reprezentowanego przez obiekt. Zdefiniuj odpowiednie konstruktory. Przyjmij, że w liściach mogą być zarówno stałe liczbowe jak i zmienne. Przyjmij, że wartości zmiennych są przechowywane np. tablicy haszującej (możesz wykorzystać tu klasy biblioteczne).

Uwaga: nie jest konieczne parsowanie wyrażeń, wyrażenia można budować np. tak:

```
wyrazenie = new Dodaj(new Stala(4), new Zmienna("x"))
```

Zaprogramuj w każdej klasie metodę `String toString()` zwracającą wyrażenie w postaci napisu.

Zadanie 3. *Zadanie to jest rozszerzeniem poprzedniego zadania.* Podobnie jak wyrażenia możemy też w postaci drzew reprezentować programy. Zaproponuj odpowiednią hierarchię klas, które będą reprezentowały

- instrukcję przypisania
- instrukcję warunkową
- instrukcję pętli
- wypisanie komunikatu (stringa) na konsolę.

Możesz przyjąć, że wyrażenia arytmetyczne można interpretować jako wyrażenia logiczne tak jak np. w języku `C`.

Jako przykład podaj jakiś niebanalny program, np. obliczenie silni.

Zadanie 4. *Zadanie to jest rozszerzeniem zadania 2.* Wyrażenia z jedną zmienna możemy traktować jak funkcje; możemy więc np. obliczać pochodne. Zaprogramuj algorytm, który dla

danego drzewa wyrażeń (będącego funkcją) zbuduje nowe drzewo reprezentujące pochodną tej funkcji. Możesz przyjąć, że algorytm nie musi sprawdzać, czy drzewo jest faktycznie funkcją.

Za każde zadanie można otrzymać do 4 pkt, jednak można oddać nie więcej niż 2 zadania. Proszę do każdego ocenianego zadania dołączyć króciutki program ilustrujący możliwości zaprogramowanych klas.

Marcin Młotkowski