23 marca 2024

Programowanie obiektowe

Lista 5.

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

Zadanie 1
Zaprogramuj kolekcję w formie listy (jedno lub dwukierunkowej) ***OrderedList*** przechowującą elementy w kolejności rosnącej wraz z metodami

- `void add_element(T elem)`
- `T get_first()`: zwraca pierwszy element listy;
- `String toString()`: zwraca string zawierający stringi elementów listy.

OrderedList musi przechowywać obiekty implementujące interfejs `Comparable<T>`.
Zaprogramuj hierarchię klas zawierającą przynajmniej cztery klasy implementujące `Comparable<T>`, na przykład hierarchię reprezentującą stopnie wojskowe czy talię kart ze starszeństwem takim jak w wybranej grze karcianej. 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.
Implementując porównanie obiektów możesz skorzystać z dostępnych w Javie *komparatorów*.

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 ***Expression***. W każdej klasie zaprogramuj metody

- `public int evaluate()` obliczająca wartość wyrażenia;
- `String toString()` zwracająca reprezentację wyrażenia.

Wymagane jest uwzględnienie co najmniej stałych, zmiennych i dwóch operacji arytmetycznych.
Uwaga 1: do obliczenia wartości wyrażenia zawierającego zmienne konieczna jest modyfikacja tej specyfikacji poprzez dodanie informacji o tym, jakie wartości są zapamiętane w zmiennych.
Uwaga 2: nie jest wymagane parsowanie wyrażeń, można je tworzyć np. tak:

```
Expression expr = new Add(new Const(4), new Variable("x"))
```

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.

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.
Algorytm powinien być zaprogramowany jako metoda klasy (i podklas) ***Expression***

```
Expression derivate()
```

która zwraca nowe drzewo wyrażeń.

Marcin Młotkowski