

Programowanie obiektowe

Lista 3.

Wymagania do implementacji

- Poniższe zadania należy wykonać nie wykorzystując klas bibliotecznych. 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. Zadania należy zaprogramować w C#.
- Każde zadanie przygotuj w dwóch plikach źródłowych: w jednym samą implementację klas, w drugim krótki przykładowy program ilustrujący wykorzystanie zbudowanej biblioteki. Implementacje klas skompiluj w postaci modułów **dll**, a następnie odwołując się tylko do modułu **dll** skompiluj plik z przykładowym użyciem tych klas do pliku wykonywalnego.

W komentarzu w jednym z plików źródłowych oprócz imienia i nazwiska umieść też polecenia jakich użyto do skompilowania plików źródłowych. W przypadku korzystania z ze środowisk typu Visual Studio bądź .NET Core SDK proszę napisać w komentarzu informację o środowisku i zamieścić na SKOSie odpowiedni plik z konfiguracją *rozwiązania* (*solution*) bądź *projektu* (*project*).

Zadanie 1

Zaprogramuj klasę **Lista<T>** implementującą metody dodawania i usuwania elementów z początku i końca listy, oraz metodę sprawdzania jej niepustości.

- **push_front(T elem), push_back(T elem)**: metody dodawania elementów z początku i końca listy;
- **pop_front(), pop_back()**: metody usuwania elementów z początku i końca listy i zwracające usuwane elementy. Jeśli lista jest pusta, zaprogramuj
- **is_empty()**

Istotne jest, aby elementy listy nie były obiektami klasy **Lista**, lecz elementami innej klasy, której polami są: pole zawierające wartość typu T, oraz odnośniki do innych elementów listy. Przyjmij taką implementację klasy Lista, aby działała ona efektywnie zarówno gdy jest wykorzystywana jako kolejka, jak i stos, tj. aby operacje dodawania i usuwania elementów na początku i koniec działały w czasie stałym. Operacja (metoda) usuwania elementu powinna zwracać jako wartość usuwany element.

Zadanie 2

Zaimplementuj klasę **MyDictionary<K, V>** przechowującą pary elementów (**key**, **value**) odpowiednio typów K, V. Pierwszym elementem pary jest klucz, a drugim wartość. Klasa powinna implementować metodę dodawania, wyszukiwania i usuwania elementu wskazywanego przez klucz.

Efektywność nie jest istotna, należy pamiętać, że klucze są unikatowe.

Zadanie 3

Na wykładzie został omówiony wzorzec **Singleton**, który pozwala na utworzenie tylko jednej instancji klasy. Zaprogramuj klasę **TimeNTon**, która będzie działała w następujący sposób:

- w godzinach pracy tworzy co najwyżej N instancji klasy, N jest ustaloną w kodzie źródłowym stałą. Przyjmij, że jeżeli zostanie utworzonych N instancji, to kolejne żądania obiektu zwrócą kolejne istniejące już instancje klasy;
- poza godzinami pracy zawsze zwracany jest ten sam obiekt. Wcześniej „wydanych” obiektów nie trzeba usuwać.

Zaprogramuj klasę w wersji leniwej.

Zadanie 4

Zaprogramuj klasę **Wektor** implementującą n wymiarowe wektory swobodne (wymiar jest zadawany w konstruktorze). Przyjmij, że współrzędne wektora są pamiętane za pomocą liczb typu `float`. Zaprogramuj operatory

- dodawania wektorów;
- iloczynu skalarnego wektorów;
- mnożenia wektora przez skalar (liczbę typu `float`);

Zaprogramuj również metodę `float norma()` zwracającą długość wektora; wykorzystaj w niej jeden z zaprogramowanych już operatorów.

Implementując to zadanie można korzystać z tablic.

Marcin Młotkowski