6 marca 2025

Programowanie obiektowe

Lista 2.

Poniższe zadania należy zaimplementować w języku $C^\sharp$. 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
Zadeklaruj klasę ***IntStream*** implementującą strumień liczb naturalnych, która implementuje publiczne metody:

```
int next();
bool eos();
void reset();
```

gdzie kolejne wywołania metody `next()` zwracają kolejne liczby naturalne począwszy od zera, wartość metody `eos()` oznacza koniec strumienia, a `reset()` inicjuje na nowo strumień. Zadeklaruj dwie podklasy klasy ***IntStream***

- ***FibStream*** implementującą strumień kolejnych liczb Fibonacciego, tj. wartościami kolejnych wywołań metody `next()` są kolejne liczby Fibonacciego.

  Oczywiście ze względu na ograniczony rozmiar typu `int` możliwe jest jedynie zwrócenie liczb Fibonacciego mniejszych niż rozmiar typu. Gdy nie jest możliwe obliczenie kolejnej liczby pierwszej, wartość `eos()` powinna być `true`. Przykład:

  ```
  FibStream fs = new FibStream();
  fs.next(); // zwraca 1
  fs.next(); // zwraca 1
  fs.next(); // zwraca 2
  fs.next(); // zwraca 3
  ```

- klasę ***RandomStream***, w której metoda `next()` zwraca liczby losowe. W takim wypadku `eos()` jest zawsze fałszywe.

Wykorzystaj te klasy do implementacji klasy ***RandomWordStream*** realizującej strumień losowych stringów o długościach równych kolejnym liczbom Fibonacciego. Przykład:

```
RandomWordStream rws = new RandomWordStream();
rws.next(); // zwraca losowy string o dł 1
rws.next(); // zwraca losowy string o dł 1
rws.next(); // zwraca losowy string o dł 2
```

Zadanie 2
Zadeklaruj w $C^\sharp$ klasę ***Array*** implementującą jednowymiarowe tablice typu `int` za pomocą list dwukierunkowych o początkowych granicach indeksów wskazywanych przez parametry konstruktora. Przyjmij, że rozmiar tablicy i jej granice indeksowania mogą być zmieniane podczas działania programu za pomocą odpowiednich metod. W implementacji zwróć uwagę na to, aby typowa operacja przeglądania tablicy taka jak

```
Array a1 = new Array(0,100);
Array a2 = new Array(0,100);
Array a3 = new Array(0,100);
...
for (int i = 0; i < 100; i++)
    a3.set(i, a1.get(i) + a2.get(i));
```

była wykonywane efektywnie, tj. bez przeglądania tablicy za każdym razem.

Zadanie 3
Zdefiniuj klasę ***LazyIntList*** implementującą leniwą listę kolejnych liczb całkowitych wraz z metodami

```
int element(int i);
```

zwracającą i-ty element listy oraz metodą

```
int size();
```

która zwraca liczbę elementów aktualnie przechowywanych w liście. Elementami tej listy są kolejne liczby całkowite. „Leniwość" takiej listy polega na tym, że na początku jest ona pusta, jednak w trakcie wywołania metody `element(100)` budowanych jest pierwszych sto elementów. Gdy dla takiej listy wywołamy metodę `element(102)` do listy dopisywane są brakujące dwa elementy. Natomiast jeśli teraz zostanie wywołana metoda `element(40)`, to ten element już jest na liście i wystarczy go odszukać i zwrócić jako wynik. Przykład:

```
lista = new LazyIntList(); // lista.size() == 0
Console.WriteLine(lista.element(40)); // lista.size() == 40
Console.WriteLine(lista.element(38)); // lista.size() == 40
```

Oczywiście, wywołanie `lista.element(40)` powinno zawsze zwrócić tę samą wartość.
Zaimplementuj klasę ***LazyPrimeList*** jako podklasę ***LazyIntList*** reprezentującą listę liczb pierwszych, tj. `element(i)` zwraca i-tą liczbę pierwszą[1].
Można korzystać z list ze standardowych bibliotek.

Zadanie 4
Słowa Fibonacciego to ciągi słów zbudowane z dwóch liter: a i b. Kolejne słowa te definiujemy następująco:

$$S_n = \begin{cases} a & \text{gdy } n = 1 \\ b & \text{gdy } n = 2 \\ S_{n-1} \bullet S_{n-2} & \text{gdy } n > 2 \end{cases}$$

gdzie $S_{n-1} \bullet S_{n-2}$ oznacza konkatenację słów S_{n-1} i S_{n-2}.
Zaprogramuj klasę ***KolejneSłowaFibonacciego***, która będzie implementować

- konstruktor bezparametrowy, który za pierwsze dwa słowa przyjmuje domyślnie `"a"` i `"b"`;
- konstruktor z dwoma parametrami `slowo1` i `slowo2`, które są pierwszym i drugim słowem Fibonacciego;
- metodę `public string next()`, która będzie za każdym razem zwracać kolejne słowo Fibonacciego.

Zaprogramuj również drugą klasę ***JakieśSłowoFibonacciego***, która będzie implementować

[1]Nie jest wymagana żadna zaawansowana implementacja sprawdzania pierwszości liczby.

- takie same konstruktory jak klasa ***KolejneSłowaFibonacciego***;
- metodę `public string slowo(int i)`, która zwraca i-te słowo Fibonacciego.

W tej drugiej klasie zadbaj, by metoda `slowo(int i)` nie liczyła za każdym razem od początku całego ciągu, tylko zapamiętywała raz policzone słowa i korzystała z nich przy kolejnych wywołaniach.
Nazwy klas ***KolejneSłowaFibonacciego*** i ***JakieśSłowoFibonacciego*** są przykładowe i mogą być zmienione.

Marcin Młotkowski