3 marca 2021

# Programowanie obiektowe

Lista 2.

Poniższe zadania należy zaimplementować w $C^\sharp$. 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***

- ***PrimeStream*** implementującą strumień liczb pierwszych, tj. wartościami kolejnych wywołań metody `next()` są kolejne liczby pierwsze. Oczywiście ze względu na ograniczony rozmiar typu `int` możliwe jest jedynie zwrócenie liczb pierwszych mniejszych niż rozmiar typu. Gdy nie jest możliwe obliczenie kolejnej liczby pierwszej, wartość `eos()` powinna być `true`. Przykład:

  ```
  PrimeStream ps = new PrimeStream();\\
  ps.next(); // zwraca 2\\
  ps.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 pierwszym. Przykład:

```
RandomWordStream rws = new RandomWordStream();
rws.next(); // zwraca losowy string o dł 2
rws.next(); // zwraca losowy string o dł 3
```

### 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**

Zaprogramuj klasę ***BigNum***, której obiekty pamiętają duże liczby całkowite (ich maksymalny rozmiar może być zadany stałą) wraz z operacjami dodawania, odejmowania, mnożenia i dzielenia całkowitego. Zaprogramuj również metodę wypisującą takie liczby na ekranie.

Wartością początkową obiektu powinna być wartość typu `int` podana jako argument w konstruktorze.

**Zadanie 4**

Zdefiniuj klasę ***ListaLeniwa*** 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 losowe 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 ListaLeniwa(); // 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 ***Pierwsze*** jako podklasę ***ListaLeniwa*** reprezentującą listę liczb pierwszych, tj. `element(i)` zwraca $i$-tą liczbę pierwszą[1].

Można korzystać z list ze standardowych bibliotek.

*Marcin Młotkowski*

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