

## 11.6 Klasa do obsługi liczb wymiernych

Klasa do obsługi liczb wymiernych, którą teraz zaprojektujemy w celu zilustrowania korzyści wynikających z programowania obiektowego, służy do zgrabnego wykonywania operacji na liczbach wymiernych przedstawianych w postaci nieskracalnego ułamka. Na przykład, wyobraźmy sobie, że musimy sprawdzić, czy nasz młodszy brat dobrze policzył wartość następującego wyrażenia:

$$\frac{5}{6} + \frac{7}{8} + \frac{1}{12} + \frac{5}{24}$$

Jak wiadomo, programista woli napisać program niż cokolwiek liczyć na papierze. Przystępując do pisania programu, warto to zrobić klarownie i tak, by można go było wykorzystywać do różnych celów. Na przykład, stosując klasę Wymierne można również wykonywać dokładne obliczenia typu:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{10}$$

czy też

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} \dots + \frac{1}{100}$$

Z sumowaniem szeregów (oczywiście skończonych) trzeba jednak uważać, ponieważ wykonując dokładne obliczenia na ułamkach, w których są duże liczby, można bardzo szybko wyjść poza zakres wykorzystywanego typu. Na przykład, wartości wyrażenia:

$$\frac{1077749}{705600} + \frac{1}{81}$$

nie da się policzyć przy stosowaniu zmiennych typu **int**. Jest to możliwe dla zmiennych typu **long**, ale i tak w trakcie dosumowywania składników szybko wyjdziemy poza zakres tego typu.

W pierwszym przybliżeniu klasę Wymierne zaprojektujemy następująco:

```
class Wymierne
{
    private long licznik, mianownik;
}
```

Do reprezentacji liczby wymiernej wykorzystamy dwa pola: *licznik* oraz *mianownik* o oczywistym znaczeniu:

$$\text{liczba} = \frac{\text{licznik}}{\text{mianownik}}$$

Zadaniem klasy będzie po wykonaniu jakiegokolwiek obliczenia skrócenia ułamka. Do tego celu zastosujemy prywatną metodę `ObliczNWP()` znajdującą największy wspólny dzielnik. A oto treść tej metody:

```
private long ObliczNWP(long a, long b)
// znajdowanie największego wspólnego dzielnika
{
    long pomoc;

    // przypadek szczególny
    if (a==0)
        return 1;

    // zamiana na liczby dodatnie
    if (a < 0)
        a = - a;
    if (b < 0)
        b = - b;

    // pierwsza liczba musi być większa
    if (a < b) {
        pomoc = a;
        a = b;
        b = pomoc;
    }

    // znajdowanie największego wspólnego dzielnika
    // metodą reszt z dzielenia całkowitego
    while ( a % b != 0) {
        pomoc = b;
        b = a % b;
        a = pomoc;
    }
    return b;
}
```

Zgodnie z założeniem po każdej operacji następuje próba skrócenia ułamka, a zatem metodę `ObliczNWP()` wywołujemy również w konstruktorze. Treść konstruktora jest następująca:

```
public Wymierne(long aLicznik, long aMianownik)
{
    if (aMianownik == 0) {
        System.out.println(
            "BŁĄD - zerowa wartość mianownika");
        aMianownik = 1;
    }
    long nwp;
    // skrócenie ułamka
    nwp = ObliczNWP(aLicznik, aMianownik);
    licznik = aLicznik/nwp;
    mianownik = aMianownik/nwp;
}
```

Warto też skonstruować drugą wersję konstruktora przewidzianą do tworzenia obiektów - liczb wymiernych o mianowniku równym 1.

```
public Wymierne(long aLicznik)
{
    licznik = aLicznik;
    mianownik = 1;
}
```

Obiekty klasy `Wymierne` możemy tworzyć w podany niżej sposób:

```
Wymierne x = new Wymierne(18,24),
y = new Wymierne(3);
```

Dla obiektu  $x$  wartość pola *licznik* po wykonaniu skrócenia będzie 3, a wartość pola *mianownik* to 4. Dla obiektu  $y$  wartość pola *licznik* wynosi 3, a wartość pola *mianownik* jest równa 1 (zapewnia to druga wersja konstruktora klasy `Wymierne`).

Możemy teraz przystąpić do zaprojektowania kilku metod ułatwiających wykorzystanie klasy. Do dodawania dwóch liczb wymiernych zastosujemy metodę `Dodaj()`. W metodzie tej skorzystamy z prostej reguły:

$$L = \frac{a}{b} + \frac{c}{d} = \frac{a * d + b * c}{b * d}$$

A oto definicja metody Dodaj():

```
public Wymierne Dodaj(Wymierne x)
{
    long nowyLicznik, nowyMianownik;

    nowyLicznik = licznik * x.mianownik +
    mianownik * x.licznik;

    nowyMianownik = mianownik * x.mianownik;

    // utworzenie obiektu klasy Wymierne
    Wymierne nowy = new Wymierne(nowyLicznik,nowyMianownik);
    // zwrócenie referencji
    return nowy;
}
```

Zwróćmy uwagę, że parametrem metody Dodaj() jest obiekt *x* klasy Wymierne. W treści metody do dyspozycji mamy zatem pola obiektu *x*: *x.licznik* i *x.mianownik* oraz pola *licznik* i *mianownik* obiektu, na rzecz którego jest wywoływana metoda Dodaj(). Skrócenie utworzonego ułamka następuje dopiero w konstruktorze przy tworzeniu nowego obiektu. Metoda zwraca referencję do nowo utworzonego obiektu zawierającego wynik sumowania dwóch liczb wymiernych.

Dla przykładu rozważmy następujące instrukcje:

```
Wymierne s; // wartością referencji s jest null
// utworzenie dwóch obiektów klasy Wymierne
Wymierne a = new Wymierne(5,6);
Wymierne b = new Wymierne(7,8);
// wykonanie dodawania s = a + b
s = a.Dodaj(b);
```

Metoda Dodaj() jest wywoływana dla obiektu *a*, gdzie parametrem formalnym jest obiekt *b*. Zwracana jest referencja do nowego obiektu, którą przypisujemy zmiennej *s*.

A oto definicje dwóch kolejnych metod Odejmij() oraz Razy(), które są zaprojektowane w podobny sposób jak metoda Dodaj().

```
public Wymierne Odejmij(Wymierne x)
```

```
{
    long nowyLicznik, nowyMianownik;

    nowyLicznik = licznik * x.mianownik -
    mianownik * x.licznik;

    nowyMianownik = mianownik * x.mianownik;
    // utworzenie obiektu klasy Wymierne
    Wymierne nowy = new Wymierne(nowyLicznik,nowyMianownik);
    // zwrócenie referencji
    return nowy;
}

public Wymierne Razy(Wymierne x)
{
    long nowyLicznik, nowyMianownik;

    nowyLicznik = licznik * x.licznik;

    nowyMianownik = mianownik * x.mianownik;

    // utworzenie obiektu klasy Wymierne
    Wymierne nowy = new Wymierne(nowyLicznik,nowyMianownik);
    // zwrócenie referencji
    return nowy;
}
```

Jeżeli chcemy dodać do siebie dwie liczby i wynik przypisać jednej z nich, to warto zaprojektować metodę `DodajDoSiebie()` o następującej treści:

```
public Wymierne DodajDoSiebie(Wymierne x)
{
    licznik = licznik * x.mianownik +
    mianownik * x.licznik;

    mianownik = mianownik * x.mianownik;

    long nwp = ObliczNWP(licznik, mianownik);
    licznik /= nwp;
```

```
    mianownik /= nwp;
    return this;
}
```

W przeciwieństwie do metody `Dodaj()` tym razem nie tworzymy nowego obiektu, lecz wynik dodawania zapisujemy w obiekcie, na rzecz którego dana metoda została wywołana. Metoda `DodajDoSiebie()` zwraca referencję do tego obiektu. Taka konstrukcja metody umożliwia na przykład potraktowanie wywołania:

```
a.DodajDoSiebie(b)
```

jako argumentu innej metody, co ilustruje poniższa instrukcja:

```
d = c.Razy(a.DodajDoSiebie(b));
```

Oczywiście, warto zdefiniować analogiczne metody dla innych operacji arytmetycznych, ale zostawiamy to już jako ćwiczenie dla Czytelnika.

Można jeszcze zaprojektować metodę `Wyprowadz()` do wyprowadzania zawartości obiektu - liczby wymiernej.

```
public void Wyprowadz()
{
    JOptionPane.showMessageDialog(null, licznik+"/"+
    mianownik, "Liczba wymierna",
    JOptionPane.INFORMATION_MESSAGE);
}
```

Można również utworzyć metodę `toString()` przesłaniającą metodę `toString()` z klasy `Object`.

```
public String toString()
{
    return licznik + "/" + mianownik;
}
```

Metoda ta jest wywoływana na przykład w poniższej instrukcji:

```
JOptionPane.showMessageDialog(null, suma,
"Liczba wymierna", JOptionPane.INFORMATION_MESSAGE);
```

W drugim parametrze metody `showMessageDialog()` powinien być podany obiekt klasy `String`, a jest podany obiekt *suma* klasy `Wymierne`. I właśnie teraz jest automatycznie wywoływana metoda `toString()`.

A oto definicja klasy `Wymierne` bez podawania treści metod omówionych wyżej.

```
import javax.swing.*;

class Wymierne
{
    private long licznik, mianownik;

    public Wymierne(long aLicznik, long aMianownik)
    {
        // treść metody jest podana wcześniej
    }

    public Wymierne(long aLicznik)
    {
        // treść metody jest podana wcześniej
    }

    private long ObliczNWP(long a, long b)
    {
        // treść metody jest podana wcześniej
    }

    public Wymierne Dodaj(Wymierne x)
    {
        // treść metody jest podana wcześniej
    }

    public Wymierne Odejmij(Wymierne x)
    {
        // treść metody jest podana wcześniej
    }

    public Wymierne Razy(Wymierne x)
    {
```

```
// treść metody jest podana wcześniej
}

public Wymierne DodajDoSiebie(Wymierne x)
{
// treść metody jest podana wcześniej
}

public void Wyprowadz()
{
// treść metody jest podana wcześniej
}

public String toString()
{
// treść metody jest podana wcześniej
}
}
```

Zastosowanie klasy Wymierne zilustrujemy w poniższej aplikacji. Obliczymy wartość wyrażenia ułamkowego podanego na początku rozdziału oraz sumę kilku składników szeregu:

$$1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} \dots + \frac{1}{100}$$

### Aplikacja 11.3

```
public
class Dokladnie {

    public static void main(String[] args)
    {
        new Dokladnie();
    }

    public Dokladnie()
    {
```

```
int i,n;
String odp;

Wymierne s;
Wymierne a = new Wymierne(5,6);
Wymierne b = new Wymierne(7,8);
Wymierne c = new Wymierne(1,12);
Wymierne d = new Wymierne(5,24);

s = a.Dodaj(b);
s.DodajDoSiebie(c);
s.DodajDoSiebie(d);
s.Wyprowadz();
// obliczenie sumy szeregu
odp = JOptionPane.showInputDialog(
"Podaj liczbę elementów");
n = Integer.parseInt(odp);

Wymierne suma = new Wymierne(1); // suma szeregu

for (i=2; i<=n; i++) {
    // utworzenie i-tego składnika
    Wymierne skladnik = new Wymierne(1,i);

    // obliczenie kwadratu składnika
    skladnik = skladnik.Razy(skladnik);

    // dodanie do sumy szeregu
    suma.DodajDoSiebie(skladnik);
}
suma.Wyprowadz();
// ilustracja wykorzystania metody toString()
JOptionPane.showMessageDialog(null, suma,
"Liczba wymierna", JOptionPane.INFORMATION_MESSAGE);

System.exit(0);
}
}
```