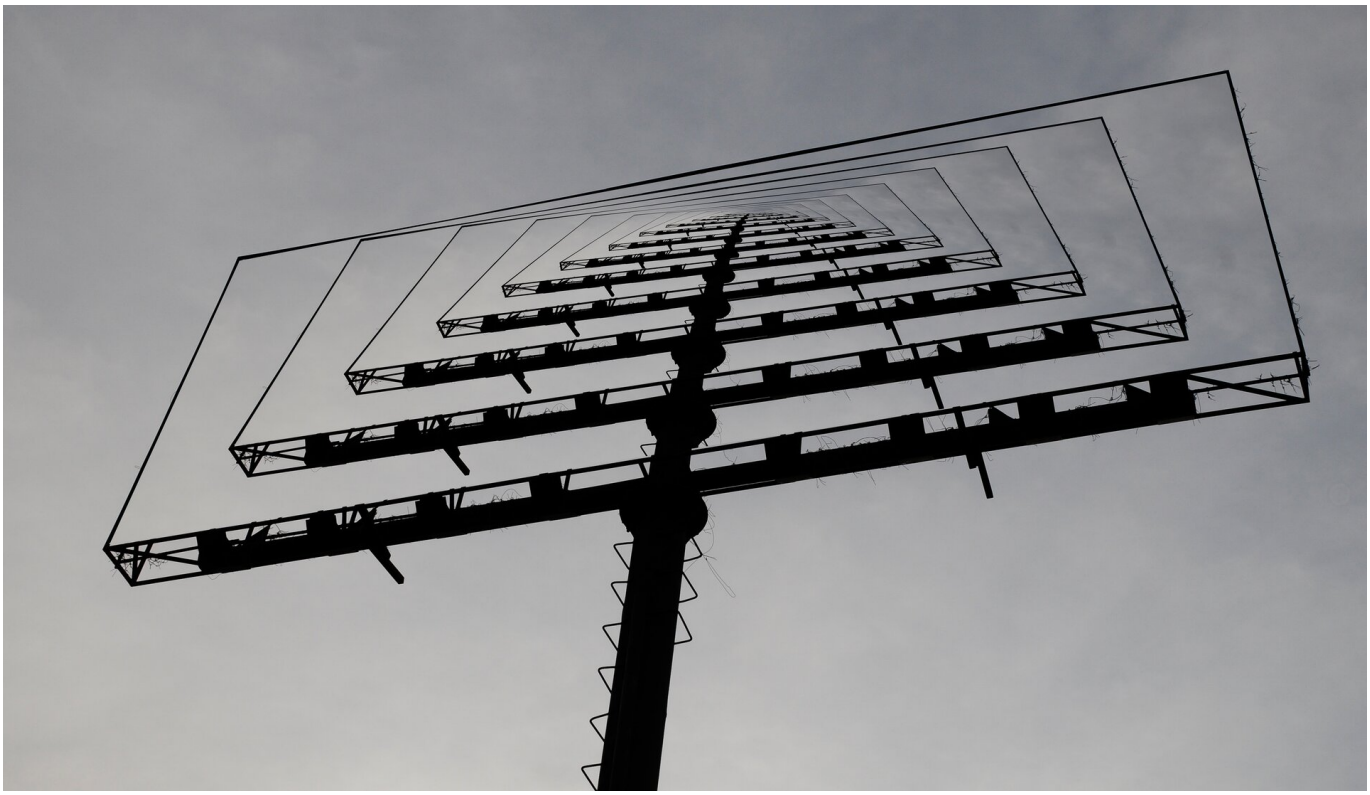
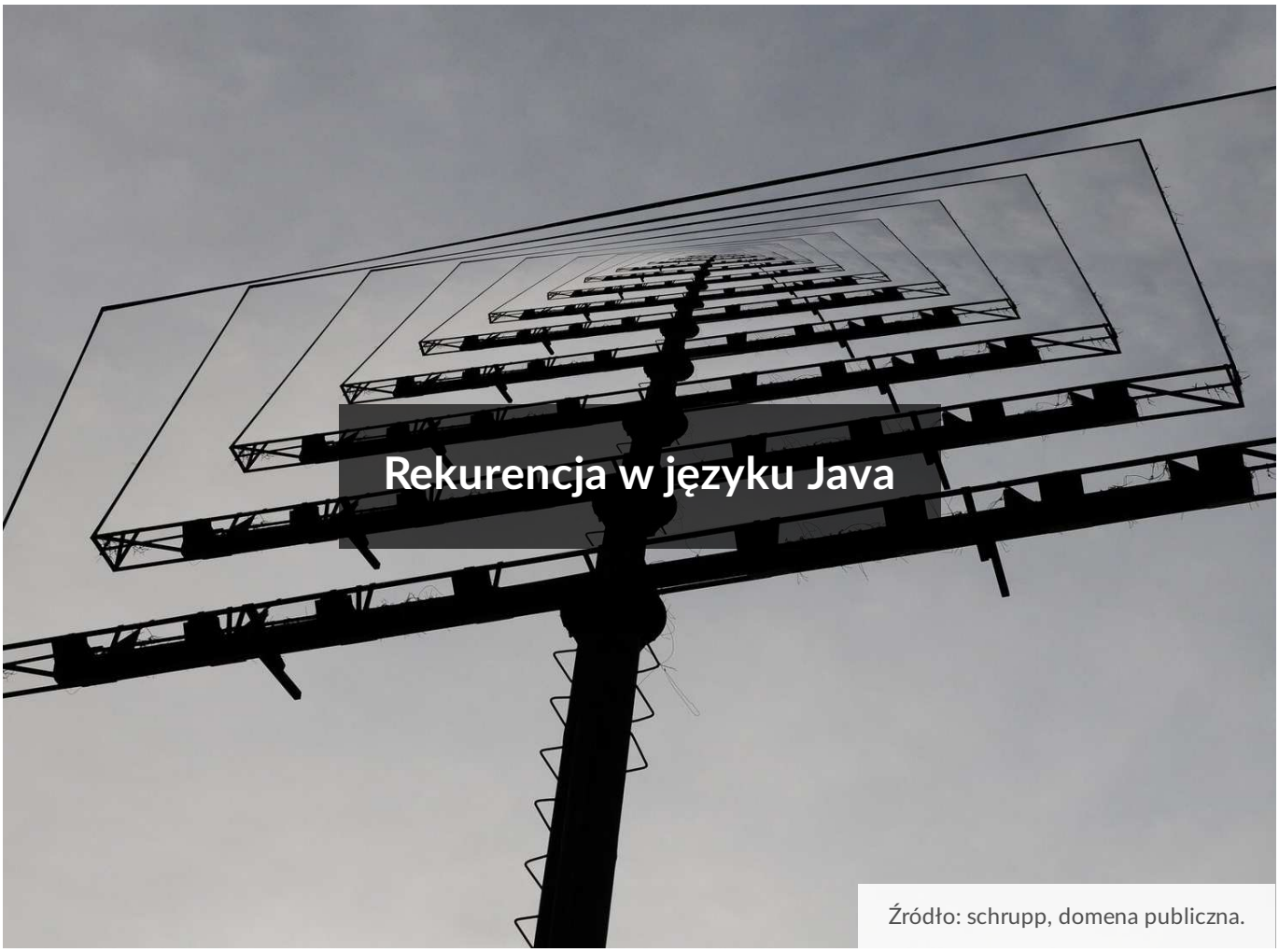


Rekurencja w języku Java



Źródło: schrupp, domena publiczna.

Rekurencja jest metodą rozwiązywania problemów, która korzysta z rozwiązań innych problemów. Jej wyjątkowość polega na tym, że problem właściwy rozwiązywany jest dla podproblemu, który powstał w wyniku podziału problemu wyjściowego.

W e-materiale Rekurencja przedstawiliśmy podstawowe informacje dotyczące omawianego zagadnienia. Kolejnym krokiem będzie implementacja tych algorytmów w wybranym języku programowania. Ten e-materiał poświęcony jest implementacji algorytmów rekurencyjnych w języku Java.

Więcej przykładów, ćwiczeń i rozwiązań znajdziesz w:

- Rekurencja – ćwiczenia,
- Rekurencja w zadaniach.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych lekcjach z tej serii:

- Rekurencja w języku C++,
- Rekurencja w języku Python.

O tym, jak zagadnienie rekurencji wyjaśnia matematyka, przeczytasz w e-materiałach:

- Ciąg określony rekurencyjnie,
- Ciąg geometryczny określony rekurencyjnie,
- Wzór ogólny ciągu określonego rekurencyjnie,
- Ciąg arytmetyczny określony wzorem rekurencyjnym.

Twoje cele

- Prześledzisz szczegółowe informacje o rekurencji.
- Przeanalizujesz przykładowe algorytmy rekurencyjne.
- Skonstruujesz w języku Java własne programy korzystające z funkcji rekurencyjnych.

Przeczytaj

W e-materiale Algorytm Euklidesa poznaliśmy algorytm służący do znajdowania największego wspólnego dzielnika dwóch liczb naturalnych. Wersję iteracyjną Algorytmu Euklidesa w języku Python omówiliśmy w e-materiale Algorytm Euklidesa w języku Java) – tym razem zbadamy podejście rekurencyjne.

Algorytm Euklidesa z odejmowaniem – implementacja algorytmu w języku Java

Zaimplementujmy z użyciem rekurencji algorytm Euklidesa w języku Java.

Nasza funkcja wygląda następująco:

```
public int NWD(int a, int b) {
    if (a != b) {
        if (a > b) {
            return NWD(a - b, b);
        }
        return NWD(a, b - a);
    }

    return a;
}
```

W funkcji `NWD()`, która przyjmuje dwie zmienne typu `int`, użylibyśmy instrukcji warunkowej `if`, w której warunek będzie taki sam, jak w przypadku pętli `while` w wykonaniu iteracyjnym, czyli `a != b`.

Jeżeli warunek jest spełniony, to znaczy `a != b`, funkcja będzie wywoływana rekurencyjnie. W przypadku gdy wartość `a` będzie większa od wartości `b`, funkcja zwróci `NWD(a - b, b)`, a w przeciwnym przypadku `NWD(a, b - a)`. W algorytmie korzystamy z równości `NWD(a, b) = NWD(a - b, b)` dla `a > b` lub `NWD(a, b) = NWD(a, b - a)` dla `b > a`, które pokazaliśmy w e-materiale Algorytm Euklidesa.

Jeżeli warunek nie jest spełniony, funkcja zwróci `a`. Oczywiście mogłaby też zwrócić `b` – jeżeli warunek nie zostanie spełniony, będzie to oznaczać, że wartość obu zmiennych jest identyczna.

Funkcja kończy swoje działanie po zwróceniu wartości, więc nie musimy stosować instrukcji `else`.

Algorytm Euklidesa z dzieleniem – implementacja algorytmu w języku C++

Kod wygląda następująco:

```
public int NWD(int a, int b) {
    if (b != 0) {
        return NWD(b, a % b);
    }
    return a;
}
```

Zadeklarujemy funkcję `NWD()` zwracającą typ `int`. Przyjmie ona parametry `a` i `b` – obydwa typu `int`. W ciele funkcji umieścimy instrukcję warunkową `if`, której testem logicznym będzie `b != 0`. Jeżeli zależność zachodzi, funkcja zwróci samą siebie z argumentami `b` i `a % b`. W przeciwnym razie funkcja zwróci zmienną `a`.

Istotne jest, żeby funkcja zwróciła przypadek podstawowy w momencie, gdy `b` przyjmie wartość `0`. W innym przypadku w następnym wywołaniu funkcji pojawiłby się argument, który doprowadziłby do wykonania operacji niedozwolonej, czyli szukania reszty z dzielenia przez `0`.

> **Ciekawostka**
>
> Co mają zatem wspólnego matrioszka i algorytmy rekurencyjne? Rosyjska lalka to w zasadzie zestaw kilku lalek – największa lalka ma w sobie trochę mniejszą, mniejsza – jeszcze mniejszą i tak dalej, aż do najmniejszej lalki, która jest tak mała, że nie da się w niej zmieścić kolejnej. Podobnie z rekurencją – algorytm rozwiązuje problem, dzieląc go na podproblemy. Robi to do momentu, w którym natrafia na przypadek podstawowy.

Słownik

> **NWD**
>
> skrót pojęcia największy wspólny dzielnik dwóch liczb całkowitych – największa liczba naturalna, która dzieli dwie rozpatrywane liczby bez reszty
>
> **przypadek podstawowy**
>
> przypadek, w którym funkcja rekurencyjna zwraca konkretną wartość, a nie wywołanie samej siebie

> **rekurencja**
>
> technika programowania polegająca na odwoływaniu się procedur lub funkcji do samych siebie, aż do momentu spełnienia warunku podstawowego

Film samouczek

Polecenie 1

Przeanalizuj prezentację dotyczącą obliczania NWD dwóch liczb naturalnych, a następnie prześledź etapy wykonywania obu funkcji rekurencyjnych dla podanych przez siebie liczb.

1

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Załóżmy, że poproszono nas o napisanie programu, który obliczy NWD dwóch podanych przez użytkownika liczb. Wprowadzona zostanie także trzecia wartość, która odpowiadać będzie za typ algorytmu Euklidesa, który ma zostać użyty do obliczenia NWD.

Jest to odpowiednio:

1. algorytm Euklidesa z dzieleniem,
2. algorytm Euklidesa z odejmowaniem.

Program ma stosować algorytm w sposób rekurencyjny.

2

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Zaczniemy od pobrania odpowiedniej biblioteki i stworzenia funkcji rekurencyjnej dla algorytmu Euklidesa z dzieleniem. Będzie ona zwracać typ `int`, a jej argumentami będą dwie zmienne typu `int`, a i b.

```
public class Main {
    public static int
NWD_mod(int a, int b)
    }
}
```

3

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Następnie umieścimy w ciele funkcji instrukcję warunkową `if`. W jej teście logicznym sprawdzimy, czy zmienna b jest różna od zera.

Jeżeli warunek zostanie spełniony, funkcja zwróci samą siebie, z argumentami b i resztą z dzielenia a przez b.

W przeciwnym razie – czyli w przypadku podstawowym – funkcja zwróci wartość zmiennej a.

```
public class Main {
    public static int
NWD_mod(int a, int b)
        if (b != 0)
            return
NWD_mod(b, a % b);
        return a;
    }
}
```

4

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Teraz zajmiemy się funkcją rekurencyjną odpowiadającą za algorytm Euklidesa

z odejmowaniem. Podobnie jak w przypadku funkcji `NWD_mod()`, funkcja ta będzie zwracać typ `int` oraz przyjmie dwie zmienne typu `int`, które nazwiemy a i b.

```
1 public static int
  NWD_mod2(int a, int b) {
2 }
```

5

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

W ciele funkcji zapiszemy instrukcję warunkową `if`, w której sprawdzimy, czy zmienne a i b mają różne wartości.

Jeżeli warunek zostanie spełniony, to w zależności od tego, która ze zmiennych przechowuje wyższą wartość, funkcja zwróci `NWD_mod2(a - b, b)` lub `NWD_mod2(a, b - a)`.

W przypadku podstawowym funkcja zwróci wartość zmiennej a.

```
1 public static int
  NWD_mod2(int a, int b) {
2     if (a != b) {
3         if (a > b)
4             return
  NWD_mod2(a - b, b);
5         return NWD_mod2(a,
  b - a);
6     }
7     return a;
8 }
```

6

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Teraz w głównej części programu zadeklarujemy zmienne, w których przechowywać będziemy podane przez użytkownika liczby oraz wybrany typ algorytmu.

```
1 public static void
  main(String[] args) {
2         int a, b, typ;
3 }
```

W celu pobrania danych wykorzystywać będziemy musieli bibliotekę `java.util.Scanner;`.

```
1 public static void
  main(String[] args) {
2         int a, b, typ;
3         Scanner scanner =
  new Scanner(System.in);
4 }
```

7

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Następnie poprosimy użytkownika o wprowadzenie trzech liczb. Znajdziemy NWD dwóch pierwszych liczb, natomiast trzecia z nich odpowiadać będzie za typ algorytmu Euklidesa.

Poprosimy użytkownika o wprowadzenie danych w następujący sposób:

```
1 public static void
  main(String[] args) {
2         int a, b, typ;
```

```
        Scanner scanner =
new Scanner(System.in);


 System.out.print("Prosze
wprowadzic dwie liczby
oraz typ: ");
        a =
scanner.nextInt();
        b =
scanner.nextInt();
        typ =
scanner.nextInt();
}
```

8

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Gdy już udało nam się uzyskać od użytkownika wszystkie potrzebne informacje, stworzymy instrukcję warunkową `if`, sprawdzimy, czy `typ` podany przez użytkownika mieści się w zbiorze {1, 2}.

W przypadku, w którym warunek ten nie będzie spełniony, zwrócimy tekst `Niepoprawny typ`.

```
public static void
main(String[] args) {
        int a, b, typ;
        Scanner scanner =
new Scanner(System.in);


 System.out.print("Prosze
wprowadzic dwie liczby
oraz typ: ");
        a =
scanner.nextInt();
```

```
7           b =
scanner.nextInt();
8           typ =
scanner.nextInt();
9           if (typ == 1 ||
typ == 2) {
10          } else {
11
 System.out.println("Niepo
prawny typ.");
12          }
13  }
```

9

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Gdy warunek `typ == 1 || typ == 2` zostanie spełniony, za pomocą kolejnych instrukcji warunkowych `if` sprawdzimy, którego algorytmu mamy użyć, aby podać odpowiedź.

Dla `typ == 1` użyjemy algorytmu Euklidesa z dzieleniem, ale dla `typ == 2` algorytmu z odejmowaniem.

```
1  public static void
main(String[] args) {
2           int a, b, typ;
3           Scanner scanner =
new Scanner(System.in);
4
5
 System.out.print("Prosze
wprowadzic dwie liczby
oraz typ: ");
6           a =
scanner.nextInt();
7           b =
scanner.nextInt();
```

```
        typ = scanner.nextInt();
        if (typ == 1 || typ == 2) {
            if (typ == 1)
                System.out.println(NWD_mod(a, b));
            if (typ == 2)
                System.out.println(NWD_mod2(a, b));
        } else {
            System.out.println("Niepoprawny typ.");
        }
}
```

10

Materiał audio dostępny pod adresem:

https://zpe.gov.pl/b/PkjrneE3r

Tym samym skończyliśmy pisać program. Kod powinien wyglądać następująco:

```
import java.util.Scanner;

public class Main {
    public static int NWD_mod(int a, int b) {
        if (b != 0)
            return NWD_mod(b, a % b);

        return a;
    }

    public static int NWD_mod2(int a, int b) {
        if (a != b) {
```

```
            if (a > b)
                return
NWD_mod2(a - b, b);
            return
NWD_mod2(a, b - a);
        }

        return a;
    }

    public static void
main(String[] args) {
        int a, b, typ;
        Scanner scanner =
new Scanner(System.in);


 System.out.print("Prosze
wprowadzic dwie liczby
oraz typ: ");
        a =
scanner.nextInt();
        b =
scanner.nextInt();
        typ =
scanner.nextInt();

        if (typ == 1 ||
typ == 2) {
            if (typ == 1)

 System.out.println(NWD_mo
d(a, b));
            if (typ == 2)

 System.out.println(NWD_mo
d2(a, b));
        } else {

 System.out.println("Niepo
prawny typ.");
        }
    }
}
```

Problem 1

Napisz program, który obliczy element ciągu o indeksie n.

Przetestuj jego działanie dla ciągu o wzorze:

$$an = \begin{cases} 1 & gdzie\ n = 0 \\ a_{n-1} + r & gdzie\ n > 0 \end{cases}$$

gdzie $a_0 = 1$, $r = 5$, $n = 3$.

Specyfikacja problemu:

Dane:

- n – liczba naturalna
- r – liczba naturalna

Wynik:

- wyraz ciągu o indeksie n

```
1
```

```
1
```

Problem 2

Napisz program, który policzy największy wspólny dzielnik (NWD) dwóch liczb całkowitych n oraz k.

Przetestuj jego działanie dla n = 24 oraz k = 28.

Rekurencyjny wzór na wyznaczanie NWD:

$$nwd\Big(k,\ n\Big) \ = \ \begin{cases} n & dla\ k=0 \\ nwd(n\ mod\ k,\ k) & dla\ k>0 \end{cases}$$

Specyfikacja problemu:

Dane:

- n – liczba naturalna
- k – liczba naturalna

Wynik:

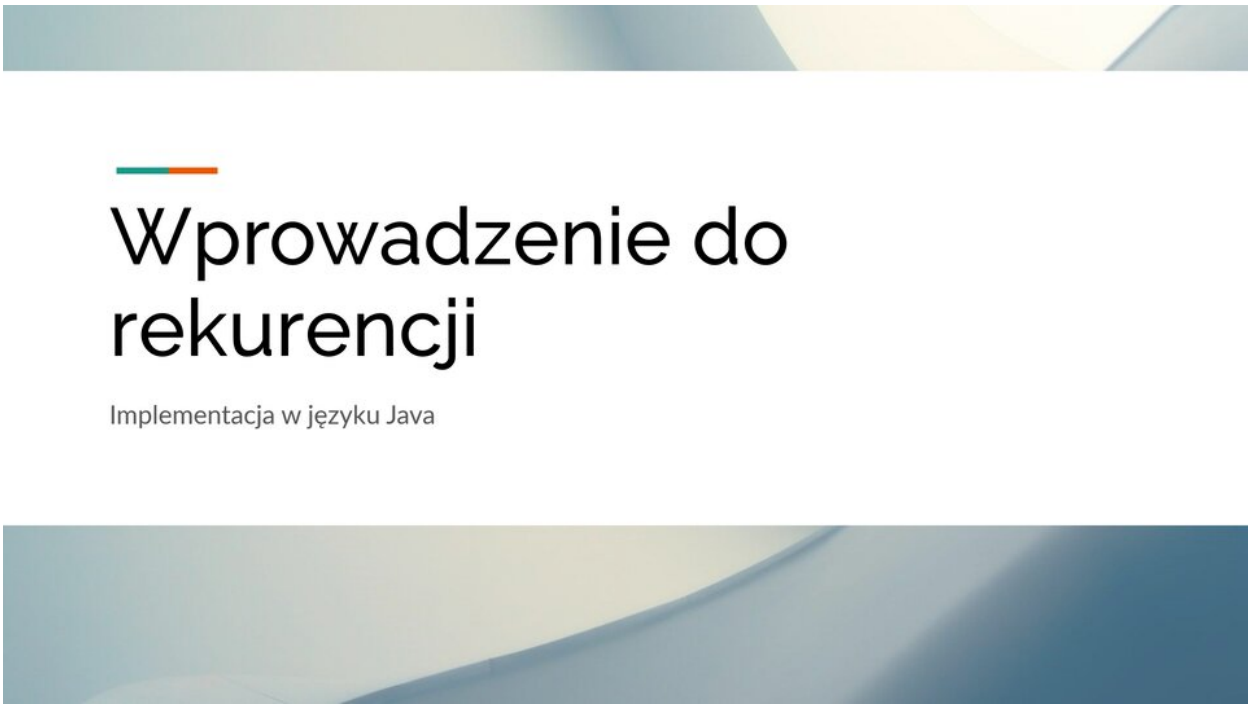
- NWD liczb n oraz k

```
1
```

```
1
```

Polecenie 2

Porównaj swoje rozwiązanie z filmem.



Film dostępny pod adresem /preview/resource/RYRfinKWvwvAx

Film nawiązujący do treści materiału

> **Polecenie 3**
>
> Zastanów się, jak wyglądałoby rozwiązanie problemu postawionego w filmie, gdyby zapisać je iteracyjnie. Jak wyglądałaby wydajność programu?

> **Polecenie 4**
>
> Zastanów się, jakie różnice dostrzegasz w zaprezentowanej implementacji w stosunku do tej, którą przedstawiono wcześniej.

> **Ważne!**
>
> W filmie mowa o tym, że dodanie poprzednich wyrazów wedle wyznaczonego wzoru jest możliwe z wykorzystaniem definicja rekurencyjnej – możemy tak zrobić w przypadku **ciągów arytmetycznych**.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który obliczy sumę n kolejnych liczb naturalnych z użyciem rekurencji. Przetestuj działanie programu dla n równego 10.

Specyfikacja problemu:

Dane:

- n – liczba naturalna

Wynik:

- liczba naturalna; suma n kolejnych liczb naturalnych

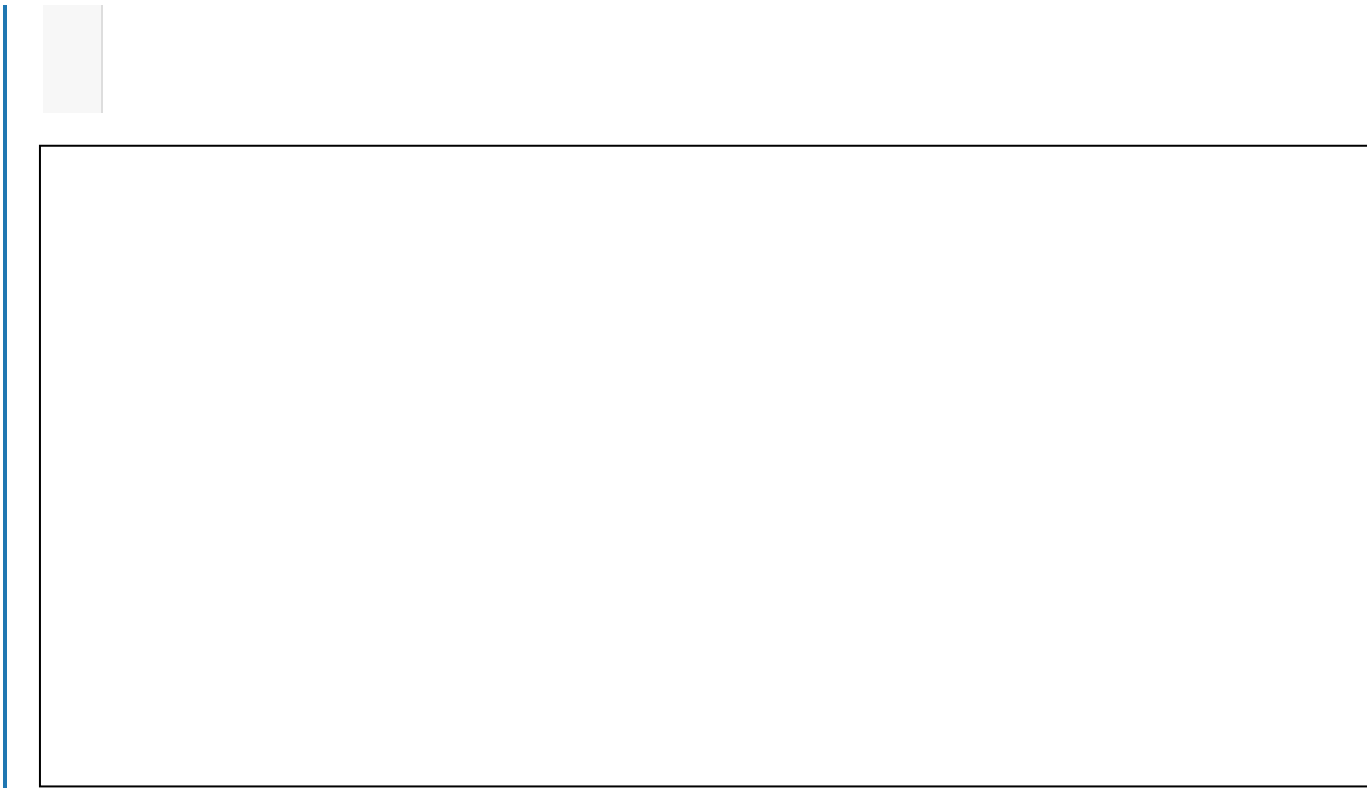
Twoje zadania

1. Program wypisuje na standardowe wyjście wartość zwracaną przez funkcję `suma` dla danych wejściowych `n = 10`.

```java
public class Main {
    static int suma(int n) {
        // Tu uzupełnij kod
    }

    public static void main(String[] args) {
        int n = 10;
        // Tu uzupełnij kod
    }
}
```

```
1
```



Ćwiczenie 2

Napisz program skracający ułamki, których elementy znajdują się w dwóch tablicach (pierwszy licznik odpowiada pierwszemu mianownikowi itd.), a następnie wypisze skróconą postać tych ułamków, np.: 1/3, 5/8 itd. Swój program przetestuj dla tablicy `liczniki = {2, 3, 4, 5, 6}` oraz `mianowniki = {4, 9, 16, 25, 36}`.

Specyfikacja problemu:

Dane:

- `liczniki` – tablica liczb całkowitych
- `mianowniki` – tablica liczb całkowitych

Wynik:

- `licznik` – liczba całkowita
- `mianownik` – liczba całkowita

Twoje zadania

1. Program skraca wartości ułamków, których liczniki i mianowniki zapisane są w dwóch tablicach, a następnie wyświetla skrócone wartości na ekranie. Licznikowi o indeksie `i` z jednej tablicy odpowiada mianownik o tym samym indeksie z drugiej tablicy.

```
1  public class Main {
2      public static void main(String [] args) {
3          int [] liczniki = {2, 3, 4, 5, 6};
4          int [] mianowniki = {4, 9, 16, 25, 36};
5
6          // tutaj dopisz kod
7          // Do wypisywania użyj tej linijki:
8          // System.out.print(licznik + "/" + mianownik + " ");
9      }
10 }
```

```
1
```

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Rekurencja w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

b) rekurencję (do generowania ciągów liczb, potęgowania, sortowania liczb, generowania fraktali),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Prześledzisz szczegółowe informacje o rekurencji.

- Przeanalizujesz przykładowe algorytmy rekurencyjne.
- Skonstruujesz w języku Java własne programy korzystające z funkcji rekurencyjnych.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda.

Przebieg lekcji

Faza wstępna:

1. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – na czym polega rekurencja?
   – jaką nazwę nosi przypadek, w którym funkcja rekurencyjna zwraca konkretną wartość, a nie wywołanie samej siebie?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie analizują przykład z sekcji „Przeczytaj” i testują zaprezentowane rozwiązania na swoim komputerze.
2. **Praca z multimedium.** Uczniowie wspólnie zapoznają się z treścią i filmem umieszczonym w sekcji „Film samouczek”. Następnie indywidualnie przechodzą do wykonywania poleceń: 1, 3 i 5. W kolejnym kroku przedstawiają swoje rezultaty osobie

z ławki. W przypadku wątpliwości i trudności przy rozwiązywaniu zadania nauczyciel omawia je na forum klasy.
3. Uczniowie w parach wykonują ćwiczenia nr 1 i 2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).

Praca domowa:

1. Uczniowie szukają problemów matematycznych i informatycznych, w których wykorzystanie rekurencji może być przydatne.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Java SE 8 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Eclipse 4.4 (lub nowszej wersji).

Wskazówki metodyczne:

- Treści w sekcji „Film samouczek" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.