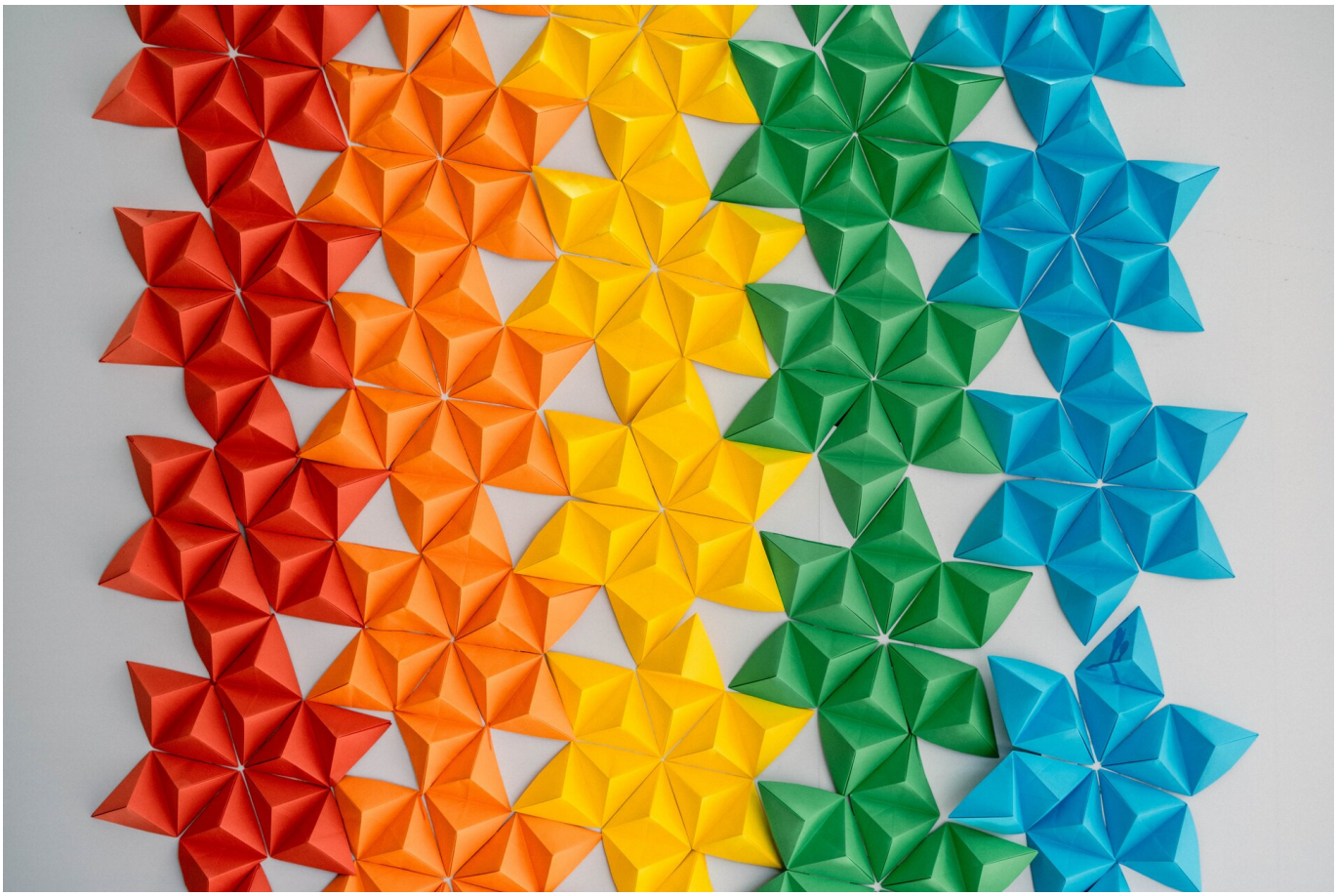


Rekurencja a iteracja w języku Java

Źródło: Chinh Le Duc, domena publiczna.

Wiesz już, czym różni się iteracja od rekurencji. W wielu sytuacjach metody te mogą być wykorzystywane zamiennie.

Bywa jednak, że rekurencja błędnie używana jest w rozwiązywaniu problemów, których natura nie jest rekurencyjna. Zrozumienie wad i zalet rekurencji wymaga umiejętności tworzenia algorytmów w sposób rekurencyjny, dlatego warto przećwiczyć rozwiązywanie problemów zarówno za pomocą rekurencji, jak i iteracji.

W e-materiale Rekurencja a iteracja porównujemy te dwie metody oraz omawiamy wady i zalety każdej z nich. Tutaj natomiast zajmiemy się analizą tego zagadnienia w języku Java.

Więcej przykładów, ćwiczeń i rozwiązań znajdziesz w Rekurencja a iteracja – zadania maturalne.

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

- Rekurencja a iteracja w języku C++,
- Rekurencja a iteracja 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

- Podsumujesz informacje na temat podstawowych właściwości iteracji i rekurencji.
- Porównasz metodę iteracyjną i rekurencyjną w języku Java.
- Rozwiążesz zadany problem metodą rekurencyjną i iteracyjną, wykorzystując język Java.

Przeczytaj

Metoda rekurencyjna – przypomnienie

Program będący zapisem metody rekurencyjnej ma postać funkcji, aby mógł odwoływać się do samego siebie. Każda funkcja rekurencyjna musi mieć **warunek zakończenia rekurencji** (zwany również **warunkiem stopu**) oraz przynajmniej jedno miejsce, w którym funkcja wywołuje samą siebie.

Kiedy ostatnie wywołanie funkcji kończy swoją pracę, również funkcja, która je spowodowała, może zakończyć swoje działanie. Proces ten powtarza się dla każdego wywołania funkcji, aż pierwsze w kolejności wywołanie funkcji zwróci wartość będącą wynikiem działania rekurencji.

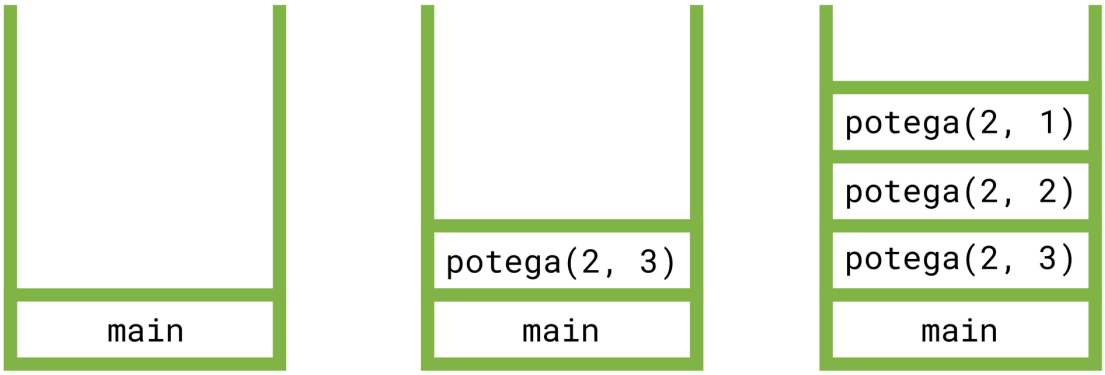
W następstwie wywołania funkcji zostaje ona odłożona na szczyt stosu (charakterystyka tej struktury danych znajduje się w e-materiale Podstawowe struktury danych: stos i kolejka), natomiast kiedy zakończy ona swoje działanie, jest ze stosu zdejmowana.

> **Ważne!**
>
> W materiale dotyczącym języka Java zamiast pojęcia *funkcja* używamy pojęcia *metoda*.

Prześledźmy, jak wygląda metoda rekurencyjna i proces tworzenia stosu z jej wykonania.

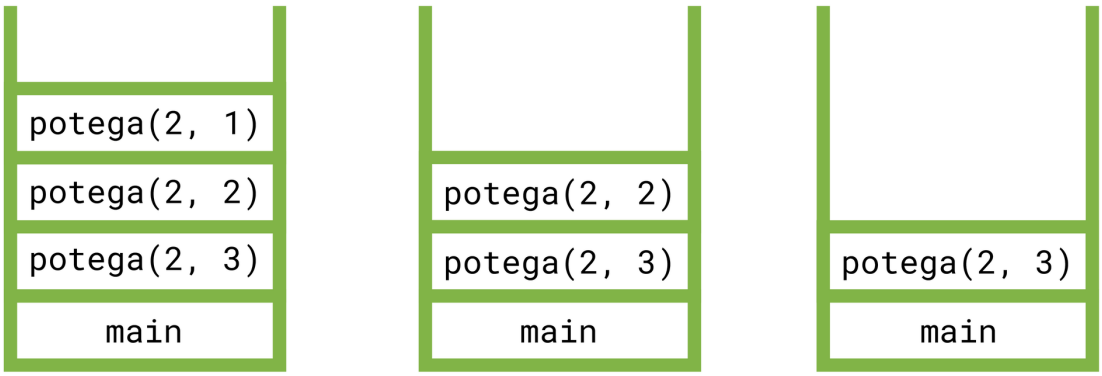
```
1  public class Main {
2      public static int potega(int a, int n){
3          if (n == 1) return a;
4          return a * potega(a, n - 1);
5      }
6
7      public static void main (String [] args) {
8          potega(2,3);
9      }
10 }
```

Najpierw wywołana zostaje metoda `main`, która jako pierwsza zostaje odłożona na stosie. Następnie wywołujemy metodę `potega(2, 3)`, a po niej metodę `potega(2, 2)`. Jako ostatnia wywołana zostaje metoda `potega(2, 1)`, w której napotkany jest warunek zakończenia rekurencji.



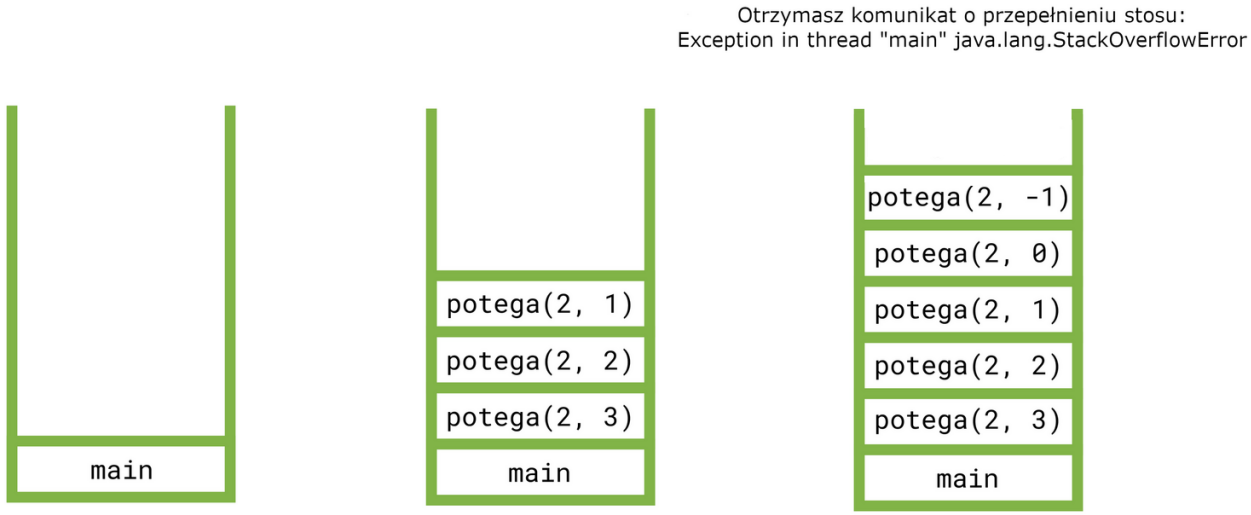
> **Ważne!**
>
> Jeżeli funkcja rekurencyjna nie będzie miała instrukcji warunkowej, która pozwoli na zakończenie jej pracy, nie wywołując przy tym kolejnej instancji funkcji, funkcja będzie rekursywnie wywoływać się w nieskończoność, podobnie jak nieskończona pętla.

Gdy wywołanie metody `potega(2, 1)` zakończy swoje działanie, zostaje ono zdjęte ze stosu, następnie przetwarzane jest wywołanie metody `potega(2, 2)`, które także jest zdejmowane ze stosu, ponieważ zwróciło swój wynik, analogiczna operacja zajdzie dla wywołania metody `potega(2, 3)`.



Kiedy liczba wywołań rekurencyjnych będzie stale rosła, np. z powodu nieumieszczenia w metodzie warunku zakończenia rekurencji, może dojść do przepełnienia stosu (o czym poinformuje komunikat:
`Exception in thread "main" java.lang.StackOverflowError`).

Jest to błąd w programowaniu polegający na zapełnieniu się roboczego obszaru pamięci operacyjnej programu wywołaniami funkcji oraz ich lokalnymi zmiennymi. Stos ma określoną pojemność, gdy zostanie przepełniony, nowe zmienne zostają umieszczone poza nim, co prowadzi do błędów. Tak wyglądałoby przepełnienie stosu dla metody `potega(2, 3)`, w której nie umieszczono warunku zakończenia rekurencji.



Źródło: Contentplus.pl Sp. z o.o., licencja: CC BY-SA 3.0.

Metoda iteracyjna – przypomnienie

Metoda iteracyjna opiera się na powtarzaniu tej samej operacji w pętli. Pętli `for` używamy, gdy wiemy, ile iteracji pętli jest potrzebnych do rozwiązania problemu. Natomiast pętlę `while` wykorzystamy, gdy nie będziemy umieli określić, ile iteracji należy wykonać w celu rozwiązania problemu. Istnieje jeszcze pętla `do while`, której działanie jest podobne do pętli `while`, różni się jednak tym, że warunek pętli sprawdzany jest dopiero po wykonaniu każdej iteracji.

Porównanie metod

Główne różnice między metodami rekurencyjną i iteracyjną to szybkość oraz sposób działania, a także – łączące się z nimi – wymagania dotyczące zasobów komputera.

W większości przypadków algorytmy rekurencyjne są mniej efektywne pamięciowo niż ich iteracyjne wersje. Różnica jest spowodowana tym, że metoda rekurencyjna wielokrotnie odwołuje się do samej siebie. Każde rekurencyjne wywołanie funkcji ma swoje własne zmienne, ponieważ wywoływana jest nowa funkcja. Każde wywołanie funkcji zajmuje

pewien obszar pamięci. Problem pojawia się wówczas, gdy tworzone jest kolejne wywołanie, a nie zostało zakończone poprzednie, a tak właśnie się dzieje przy wykorzystaniu rekurencji. Nie jest to zauważalne w przypadku mniej skomplikowanych programów, w których wywołań rekurencyjnych jest niewiele. Jednak jeśli problem jest bardziej wymagający, wykorzystanie pamięci rośnie. W momencie, gdy nieumiejętnie zastosujemy rekurencję, może dojść do przepełnienia stosu i pojawienia się komunikatu: `Exception in thread "main" java.lang.StackOverflowError`. Zaletą zastosowania algorytmu rekurencyjnego jest przejrzystość oraz czytelność kodu.

Częstym błędem, przez który rekurencyjne wersje algorytmów działają wolniej niż ich iteracyjne odpowiedniki, są nieoptymalnie skonstruowane wywołania. Jest tak w wypadku algorytmu, w którym funkcja kilka razy liczy to samo, np. algorytmu rekurencyjnego wyznaczającego elementy ciągu Fibonacciego.

Warto wiedzieć, kiedy należy użyć metody rekurencyjnej, a kiedy lepiej pozostać przy iteracyjnej. Rekurencja często ułatwia zrozumienie i rozwiązanie problemu, jeśli faktycznie jest to zgodne z jego charakterem.

Dobrym przykładem odstępstwa metody iteracyjnej w kwestii tworzenia algorytmów jest problem potęgowania. Standardową definicję potęgowania możemy zapisać następująco:

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n}$$

$$a^n = \begin{cases} a & \text{gdy } n = 1 \\ a \cdot a^{n-1} & \text{gdy } n > 1 \end{cases}$$

a – podstawa potęgi

n – liczba całkowita wykładnik potęgi

Zaprezentowana definicja nie posiada żadnej istotnej zalety względem wersji iteracyjnej. Można powiedzieć, że utrudnia zrozumienie natury problemu. Dodatkowo jej implementacja wprowadza sporo niewystępujących w przypadku wersji iteracyjnej problemów, związanych na przykład z przepełnieniem stosu.

Implementacja potęgowania liczb

Aby omówić różnice programistyczne w przypadku różnych podejść do problemu, porównajmy przykładowe implementacje metod realizujących definicje

matematyczne w języku Java. Zacznijmy od potęgowania liczb.

Specyfikacja problemu:

Dane:

- `a` – liczba naturalna; podstawa potęgi
- `n` – liczba całkowita; wykładnik potęgi

Wynik:

Program zwraca wartość zmiennej `a` podniesionej do potęgi `n`.

Wersja iteracyjna:

```
public static int potegowanieIteracyjnie(int a, int n) {
    int wynik = a;
    for (int i = 1; i < n; ++i) {
        wynik *= a;
    }

    return wynik;
}
```

Wersja rekurencyjna:

```
public static int potegowanieRekurencyjnie(int a, int n) {
    if (n == 1) return a;
    return a * potegowanieRekurencyjnie(a, n - 1);
}
```

Widoczna na pierwszy rzut oka różnica, która przemawia na korzyść rozwiązania rekurencyjnego, to długość kodu. Zauważmy, że zapis metody `potegowanieRekurencyjnie()` jest krótszy. W wielu algorytmach funkcje rekurencyjne są prostsze i krótsze w zapisie.

Porównywanie długości kodu to często jednak błąd. Liczba linijek kodu jest mniej ważna od jego wydajności. Przeanalizujmy zatem aspekty wydajnościowe obu rozwiązań. Złożoność czasowa obu funkcji jest taka sama. Inaczej sprawa ma się w przypadku złożoności pamięciowej. W wersji iteracyjnej, aby obliczyć wartość funkcji dla danego parametru `n`, deklarujemy tylko cztery zmienne. W przypadku funkcji napisanej rekurencyjnie liczba zmiennych jest znacząco zwiększona. Przy obliczaniu wartości potęgi dla danego

parametru n, będziemy mieli liczbę zmiennych rzędu $2 \cdot n$, ponieważ dla każdego rekurencyjnego wywołania funkcji tworzymy dwie zmienne lokalne a i n.

Niepotrzebnie zwiększone jest zatem zapotrzebowanie na pamięć na stosie, co może powodować niemożliwość wykonania funkcji dla dużych parametrów n. W przypadku operacji potęgowania mamy do czynienia z bardzo szybkim wzrostem wyniku. Prawdopodobnie nie doświadczymy w tym wypadku problemu z przepełnieniem stosu, gdyż i tak podniesienie liczby do odpowiednio dużej potęgi byłoby niemożliwe z uwagi na pojemność zmiennej typu `int` czy nawet `long`. Inaczej byłoby w przypadku próby obliczenia następującego działania:

$$2^{10000000} \bmod 270$$

Uzyskanie wyniku przedstawionej operacji byłoby niemożliwe do osiągnięcia z pomocą podejścia rekurencyjnego zbudowanego w sposób analogiczny do zdefiniowanej wcześniej funkcji. W tym wypadku trzeba zastosować podejście iteracyjne lub algorytm tzw. szybkiego potęgowania (w wersji zarówno iteracyjnej, jak i rekurencyjnej).

Implementacja silni matematycznej

Porównajmy teraz dwie wersje algorytmu obliczającego silnię matematyczną zapisane w języku Java.

Silnia to jednoargumentowa funkcja matematyczna, której wynikiem jest iloczyn wszystkich liczb naturalnych (oprócz zera) nie większych niż argument silni.

> **Przykład 1**
>
> $$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

Rozpiszmy przykładową postać algorytmów. Zaczniemy od metody iteracyjnej.

Specyfikacja problemu:

Dane:

- n – liczba całkowita; argument silni

Wynik:

Program zwraca wartość silni n.

```
1 public class Main {
```

```
    public static int silniaIteracyjna(int n){
        int wynik = 1;

        while(n > 0) {
            wynik = wynik * n;
            n--;
        }
        return wynik;

    }
    public static void main (String [] args) {
       System.out.print(silniaIteracyjna(5));
    }

}
```

Zmienna `n` przechowuje argument metody `silnia`, w tym przypadku jest to 5, więc matematyczna reprezentacja problemu wygląda następująco: $5!$

Specjalnym przykładem silni jest liczba $0!$, gdyż jako wynik zwraca ona 1. Dlatego do zmiennej `wynik` przypisywana jest wartość 1. W pętli najpierw sprawdzany jest warunek czy $n > 0$. Pozwala to na mnożenie zmiennej `wynik` przez kolejne wartości n. Po zakończeniu pętli wynik działania $5!$ zostaje zwrócony z metody `silniaIteracyjna`, a następnie wypisany w konsoli.

W przypadku rekurencyjnej wersji problemu, kod wygląda następująco:

```
public class Main {
    public static int silniaRekurencyjna(int n) {
        if (n == 0) return 1;
        return n * silniaRekurencyjna(n - 1);
    }

    public static void main (String [] args) {
       System.out.print(silniaRekurencyjna(5));
    }

}
```

Ponownie rozważamy przypadek specjalny dla $0!$, więc gdy `n = 0` zwracamy wartość 1. Gdy `n` jest różne od 0, wywołujemy metodę `silniaRekurencyjna(n-1)` oraz mnożymy ją

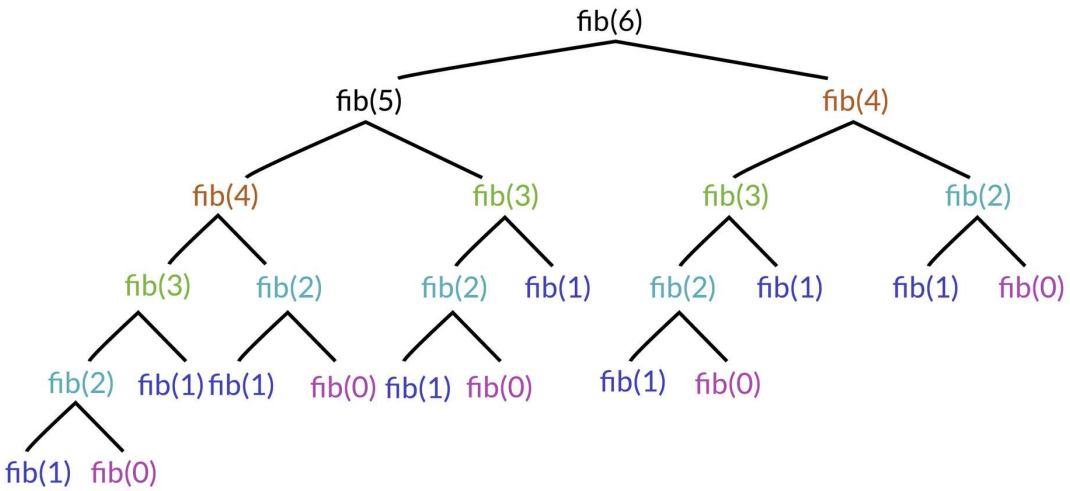
razy n.

Rekurencyjne powielanie obliczeń

Wspomnieliśmy już o tym, że nieumiejętna implementacja funkcji rekurencyjnej może skutkować zwiększoną liczbą wykonanych operacji. Zagadnienie to dobrze obrazuje problem obliczania n-tego wyrazu ciągu Fibonacciego, kod w języku Java reprezentujący rekurencyjne rozwiązanie problemu wygląda następująco:

```
1  public class Main{
2      public static int fib(int n){ //rekursywnie
3          //warunki zakończenia
4          if(n == 0) return 0;
5          if(n == 1) return 1;
6          //wywołanie rekurencyjne
7          return  fib(n - 2) + fib(n - 1);
8      }
9
10     public static void main (String [] args) {
11         System.out.println(fib(8));
12     }
13 }
```

Dla wywołania metody `fib(8)` otrzymamy następujące drzewo wywołań rekurencyjnych:



Na grafice kolorami zaznaczone są wielokrotne wywołania metody `fib(6)`, `fib(5)`, `fib(4)` itd. Przez nieumiejętne użycie rekurencji wielokrotnie obliczamy wartości identycznych wywołań metody, co powoduje ogromny wzrost zajmowanej pamięci na stosie.

Na przytoczonym przykładzie możemy zauważyć, że używanie rekurencji może czasem prowadzić do niepotrzebnego powielania obliczeń, w przypadku tego problemu opłaca się zastosować metodę iteracyjną.

```
1  public class Main{
2      public static int fib(int n){ //iteracyjnie
3          int a = 1; //pierwszy i drugi wyraz
4          int b = 1; //ciągu Fibbonaciego
5          int wyraz = 1; //wartość bierzącego wyrazu ciągu
6
7          for(int i = 3; i <= n; i++){ //pierwsze 2 wyrazy
8              wyraz = a + b;           // są już obliczone
9              a = b;
10             b = wyraz;
11         }
12         return wyraz;
13     }
14
15     public static void main (String [] args) {
16         System.out.println(fib(1));
17     }
18 }
```

Dzięki metodzie iteracyjnej używamy tylko 4 zmiennych, a także nie powielamy obliczeń.

Słownik

> **void**
>
> metoda, która nie przekazuje informacji zwrotnej, nie zwraca danych, nie przyjmuje żadnych parametrów
> **nieskończona pętla**
>
> pętla, w której nigdy nie zajdzie warunek zakończenia pętli, a polecenia w niej zawarte będą się wykonywać w nieskończoność
> **przepełnienie stosu**

błąd w programowaniu polegający na zapełnieniu się roboczego obszaru pamięci operacyjnej programu; problem dotyczy organizacji pamięci; lokalne zmienne tworzone w programie zapisywane są na stosie, który ma określoną pojemność; gdy zmienne przestają być używane, pamięć jest sukcesywnie zwalniana; w momencie, gdy zaczynamy tworzyć zmienne lokalne lub wywołania funkcji w niekontrolowanym tempie, dochodzi do zapełnienia obszaru stosu, przez co następne zmienne zostają umieszczone poza stosem

rekursja

alternatywne określenie rekurencji

Animacja

Polecenie 1

Zapoznaj się z filmem przedstawiającym programy obliczające silnię zadanej liczby z wykorzystaniem rekurencji oraz iteracji.

Film dostępny pod adresem /preview/resource/R1OjLaYbTgQQO

Film nawiązujący do treści materiału

Plik o rozmiarze 491.00 B w języku polskim

Polecenie 2

Napisz dwa programy, których zadaniem będzie obliczenie wartości n-tego elementu ciągu zdefiniowanego wzorem:

$$a_n \begin{cases} 0 & \text{gdy } n = 0 \\ 1 & \text{gdy } n = 1 \\ a_{n-1} \cdot a_{n-2} + 3 & \text{gdy } n > 1 \end{cases}$$

W pierwszym wykorzystaj rekurencję, w drugim – iterację.

Specyfikacja problemu:

Dane:

- n – liczba naturalna

Wynik:

Program na wyjściu standardowym zwróci wartość n-tego elementu ciągu.

Metoda rekurencyjna:

```java
public class Zadanie {
    public static int rekurencyjnie(int n) {
        // tutaj dopisz kod
    }

    public static void main(String[] args) {
        int n = 5; // dla testów zmień liczbę

        System.out.println(rekurencyjnie(n));
    }
}
```

```
1
```

Metoda iteracyjna:

```java
public class Zadanie {
    public static int iteracyjnie(int n) {
        // tutaj dopisz kod
    }

    public static void main(String[] args) {
        int n = 5; // dla testów zmień liczbę

        System.out.println(iteracyjnie(n));
    }
}
```

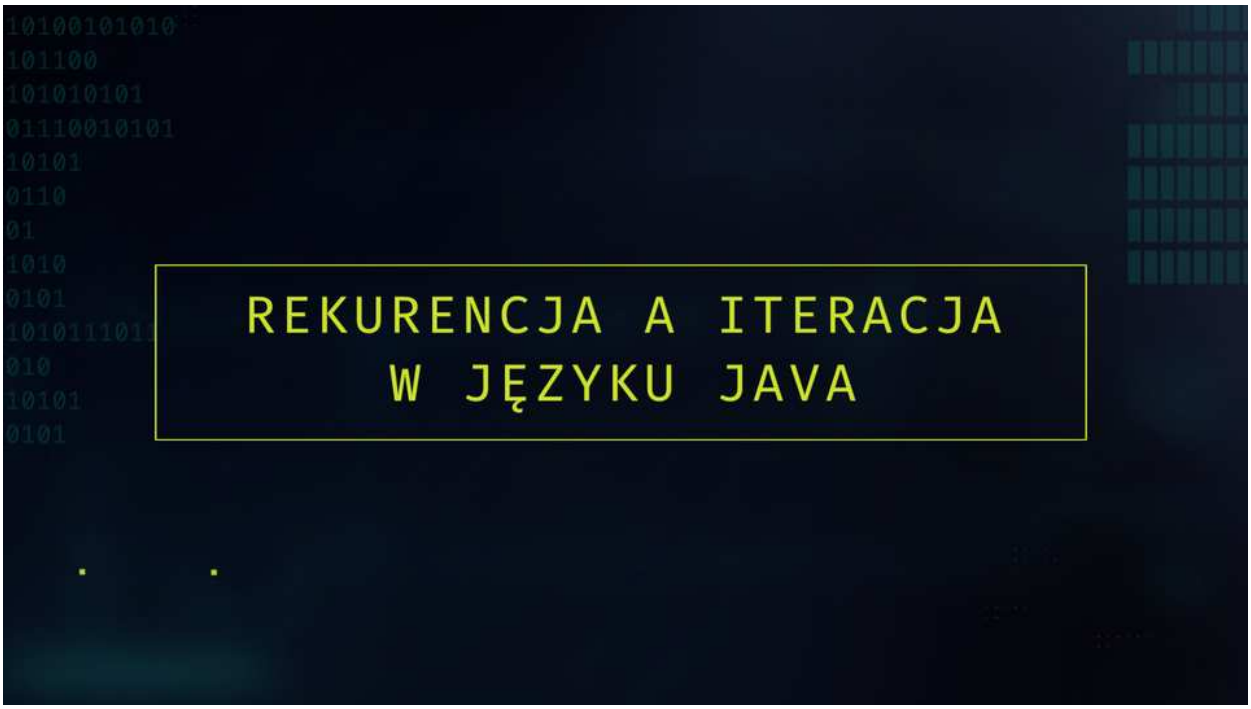
```
1
```

Polecenie 3

Określ, jaka jest złożoność obliczeniowa obu algorytmów.

Polecenie 4

Porównaj swoje rozwiązanie z animacją.



Film dostępny pod adresem /preview/resource/RHNUNdQyAyMpG

Film nawiązujący do treści materiału

Plik o rozmiarze 689.00 B w języku polskim

Polecenie 5

Na podstawie e-materiału przygotuj notatkę, w której podsumujesz najważniejsze informacje dotyczące rekurencji oraz iteracji.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który metodą iteracyjną wypisze sumę cyfr podanej liczby. Poniżej znajduje się przykładowa implementacja wykonana metodą rekurencyjną. Przetestuj swój program dla liczby 473856394.

```java
public class Main {
    public static void main(String[] args) {
        int n = 473856394;

        if (n < 0) {
            n = -1 * n;
        }

        int sumaCyfr = sumujCyfry(n);

        System.out.println(sumaCyfr);
    }

    public static int sumujCyfry(int n) {
        if (n == 0) {
            return 0;
        } else {
            return n % 10 + sumujCyfry(n / 10);
        }
    }
}
```

Specyfikacja problemu:

Dane:

- n – liczba całkowita, której sumę cyfr należy obliczyć

Wynik:

- suma – liczba naturalna, suma cyfr liczby n

Twoje zadania

1. Program wypisuje sumę cyfr danej liczby, wykorzystując iterację.

Ćwiczenie 2

Napisz program, który metodą iteracyjną wypisze wszystkie liczby naturalne nie większe od podanej liczby n i podzielne przez x (nie uwzględniając zera, w kolejności malejącej). Przetestuj go dla n = 18 oraz x = 3.

Specyfikacja problemu:

Dane:

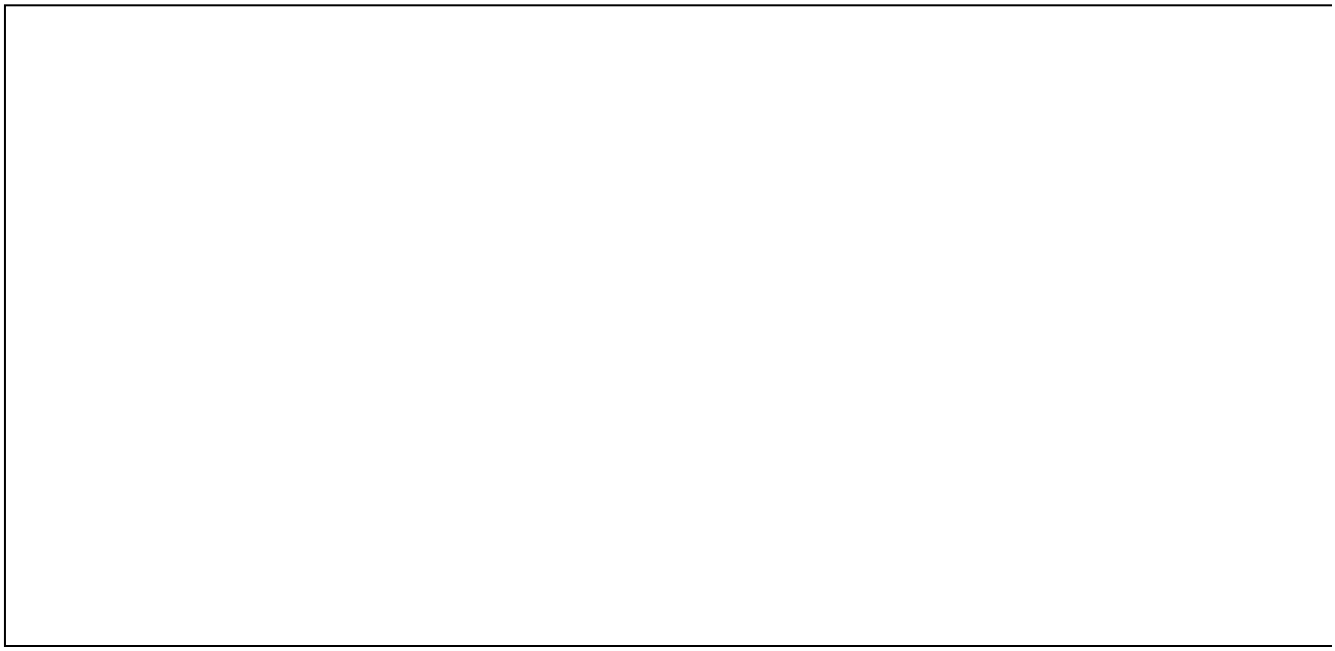
- n – liczba naturalna
- x – liczba naturalna dodatnia

Wynik:

- wypisze wszystkie liczby naturalne nie większe od podanej liczby n i podzielne przez x (nie uwzględniając zera, w kolejności malejącej)

Twoje zadania

1. Program za pomocą iteracji wypisuje wszystkie liczby naturalne podzielne przez x, które są nie większe niż podana liczba n (nie uwzględniając zera).



Ćwiczenie 3

Napisz program, który metodą rekurencyjną wypisze wszystkie liczby naturalne nie większe od podanej liczby n i podzielne przez x (nie uwzględniając zera, w kolejności malejącej). Przetestuj go dla n = 18 oraz x = 3.

Specyfikacja problemu:

Dane:

- n – liczba naturalna
- x – liczba naturalna dodatnia

Wynik:

- wszystkie liczby naturalne nie większe od podanej liczby n i podzielne przez x (nie uwzględniając zera, w kolejności malejącej)

Twoje zadania

1. Program za pomocą rekurencji wypisuje wszystkie liczby naturalne podzielne przez x, które są nie większe niż podana liczba n (nie uwzględniając zera).

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Rekurencja a iteracja w języku Java

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i 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. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

e) obliczania wartości elementów ciągu metodą iteracyjną i rekurencyjną, w tym wartości elementów ciągu Fibonacciego.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

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

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

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):

- Podsumujesz informacje na temat podstawowych właściwości iteracji i rekurencji.
- Porównasz metodę iteracyjną i rekurencyjną w języku Java.
- Rozwiążesz zadany problem metodą rekurencyjną i iteracyjną, wykorzystując język Java.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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;
- oprogramowanie dla języka Java SE 8 (lub nowszej wersji), w tym Eclipse 4.4 (lub nowszej wersji).

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Rekurencja a iteracja w języku Java". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Metodą burzy mózgów uczniowie referują najważniejsze informacje, jakie znaleźli w sekcji „Przeczytaj", przygotowując się do lekcji. Następnie chętna lub wybrana osoba podsumowuje najważniejsze różnice między iteracją a rekurencją. W razie potrzeby nauczyciel uzupełnia wypowiedź.
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Animacja". Uczniowie wspólnie zapoznają się z treścią filmu i dzielą się swoimi spostrzeżeniami na forum klasy. W kolejnym kroku w parach implementują dwa programy w języku Java. W pierwszym wykorzystują metodę rekurencyjną, w drugim – iteracyjną. Chętna lub wybrana osoba przedstawia swoje rozwiązanie na forum klasy. Nauczyciel komentuje je. W następnym kroku uczniowie porównują swoje rozwiązania z przedstawionymi w animacji.
3. W kolejnym etapie uczniowie dobierają się w pary i wykonują ćwiczenia nr 1-2 z sekcji „Sprawdź się". Następnie konsultują swoje rozwiązania z inną parą uczniów i ustalają jedną wersję odpowiedzi.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Java.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.

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 „Przeczytaj” można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.