Programowanie 1

Mateusz Duraj

Agenda

- Struktury danych Listy, Stosy i Kolejki
- Algorytmy i pseudokod
- Złożoność obliczeniowa i O-notacja
- Sortowanie na przykładzie prostych algorytmów

Zanim zaczniemy...

Pobierz repozytorium wskazane przez prowadzącego i otwórz plik :

INTRODUCTION.md

Pseudokod

- Używany jest w ujęciu algorytmów, nie jest to konkretny przypadek żadnego języka programowania
- Pseudokod "imituje kod"
- Pozwala przedstawić sposób działania funkcji/metody czy nawet całego
- W przypadku pseudokodu nie ma ustalonej strukury - występuje tutaj pewna dowolność , chodzi o przedstawienie rozwiązania problemu, w czytelny sposób

Zanim zaczniemy – pseudokod

Dla przykładu pseudokod dla wyznaczenia elementu maksymalnego w tablicy liczb całkowitych (**w javie int[] tab**) o rozmiarze **n**:

```
SET max to tab[0]
For i = 1 to n - 1
   IF  tab[i] > max:
        SET max = tab[i]
End for
Return max
```

Zadanie

Zapoznaj się z klasami **CollectionUtils1Test**, oraz **CollectionUtils2Test**

Powyższe klasy to testy dla klasy CollectionUtils w której należy zaimplementować metody (TODO).

Warunkiem wykonania poprawnie zadania jest przejście wszystkich testów.

Dodatkowo dokończ implementacje metod w klasie **`MultidimensionalArraysSolutions`**

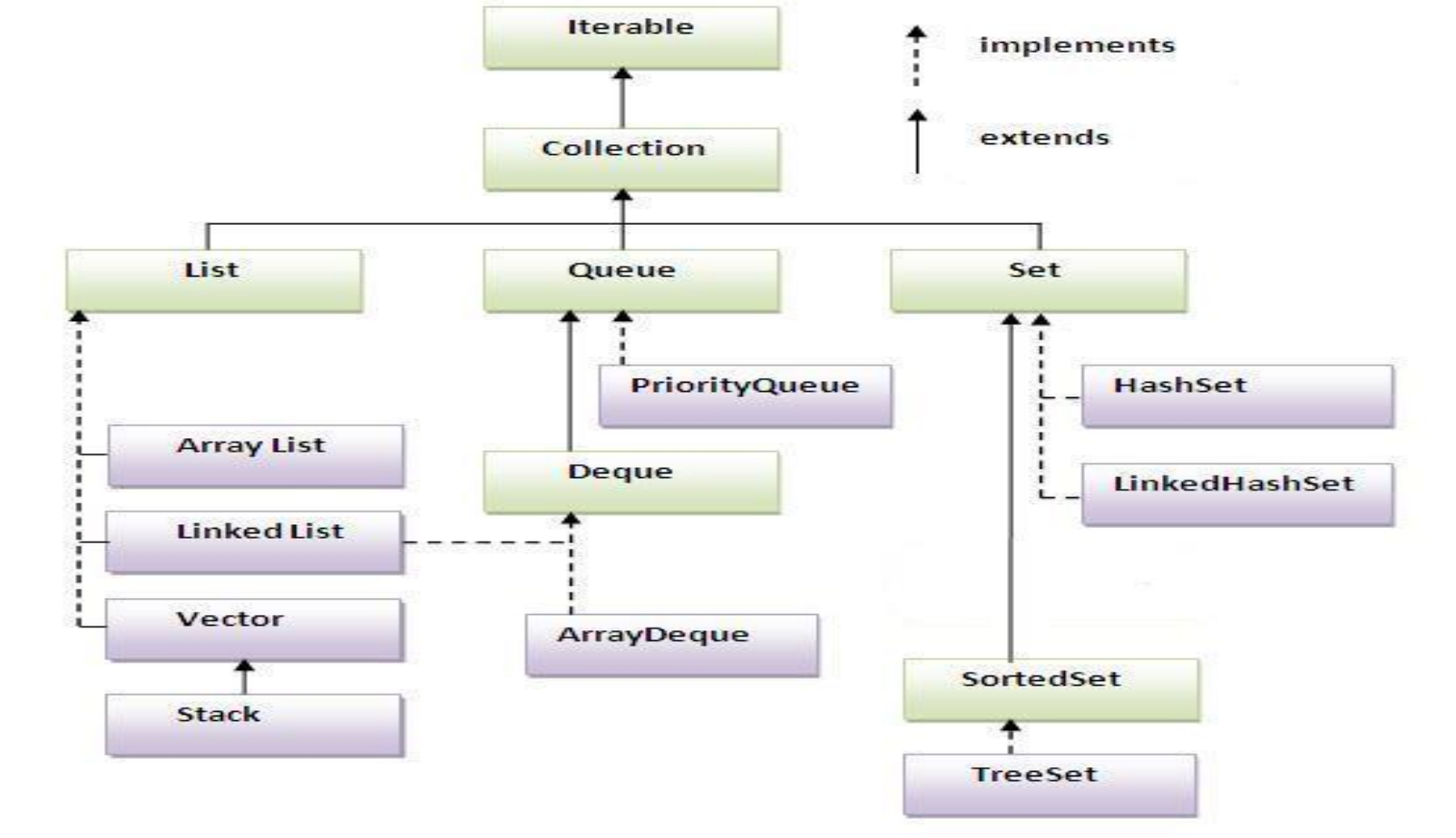
Struktury danych

Struktura danych (ang. data structure) - sposób przechowywania danych w pamięci komputera. Na strukturach danych operują algorytmy. Każda struktura danych ma charakterystyczne dla siebie właściwości. Na przykład dodanie elemntu na początek tablicy ma złożoność obliczeniową **O(n)**

Przykładowe struktury danych:

- Tablica
- Lista
- Stos
- Kolejka
- Drzewo i jego odmiany(np. Drzewo binarne)
- Graf

Struktury danych - hierarchia JCF (Java Collection Framework)



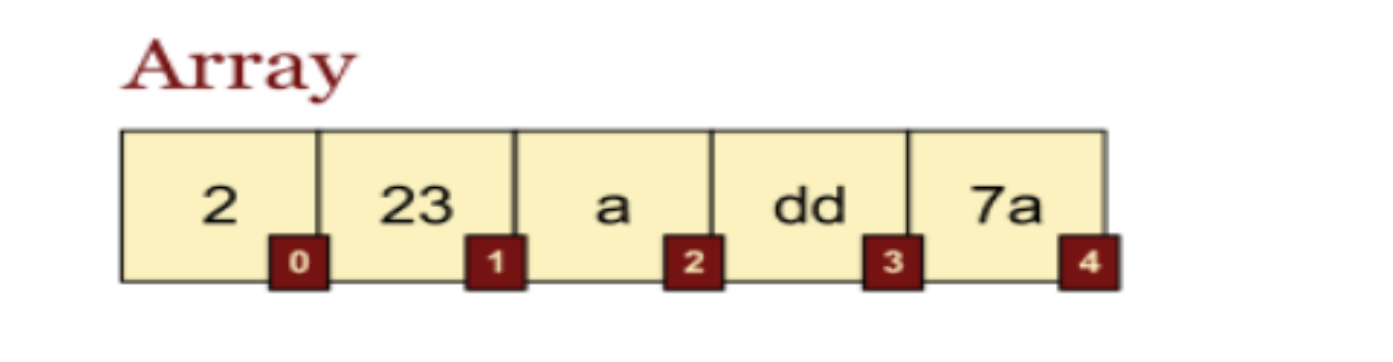
Zródło: http://javaessential.com/java-tut/images/collectionhierarchy.JPG

Struktury danych - Pobierz pdf z linku

Collection class	Thread-safe alternative	Your data				Operations on your collections						
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	Order of iteration			Performant 'contains' check	Random access		
						FIFO	Sorted	LIFO		By key	By value	By index
HashMap	ConcurrentHashMap	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
HashBiMap (Guava)	Maps.synchronizedBiMap (new HashBiMap())	✗	✓	✗	✗	✗	✗	✗	✓	✓	✓	✗
ArrayListMultimap (Guava)	Maps.synchronizedMultiMap (new ArrayListMultimap())	✗	✓	✓	✗	✗	✗	✗	✓	✓	✗	✗
LinkedHashMap	Collections.synchronizedMap (new LinkedHashMap())	✗	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
TreeMap	ConcurrentSkipListMap	✗	✓	✗	✗	✗	✓	✗	✓*	✓*	✗	✗
Int2IntMap (Fastutil)		✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	✓
ArrayList	CopyOnWriteArrayList	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓
HashSet	Collections.newSetFromMap (new ConcurrentHashMap<>())	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗
IntArrayList (Fastutil)		✓	✗	✓	✓	✓	✗	✓	✗	✗	✗	✓
PriorityQueue	PriorityBlockingQueue	✓	✗	✓	✗	✗	✓**	✗	✗	✗	✗	✗
ArrayDeque	ArrayBlockingQueue	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

ArrayList

Klasa ArrayList to dynamiczna struktura tablicy, zaimplementowana z użyciem tablicy



Autor:

ArrayList - Zadanie

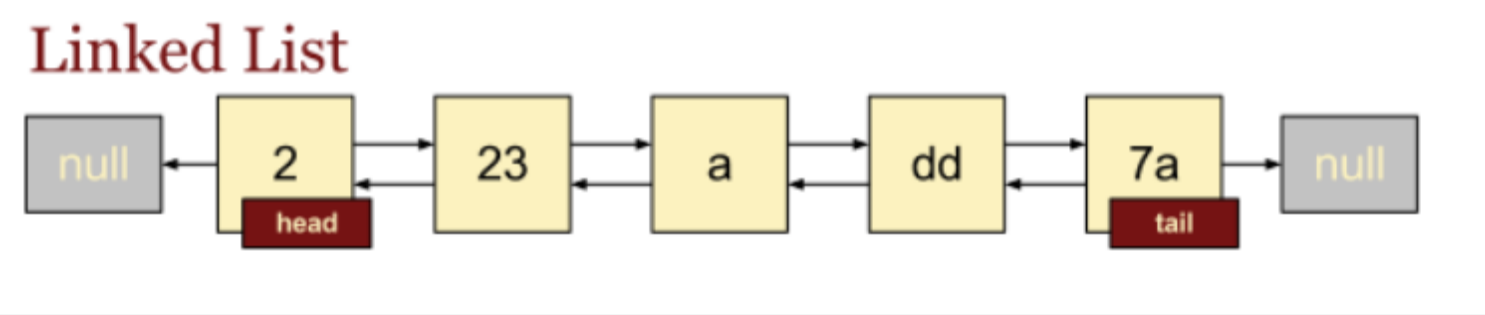
Przejdz do klasy **MyArrayList.java**

W klasie **MyArrayListTest.java** wykonaj operacje doprowadzając do przyjścia testów jednostkowych - uzupełniając implementację MyArrayList.java

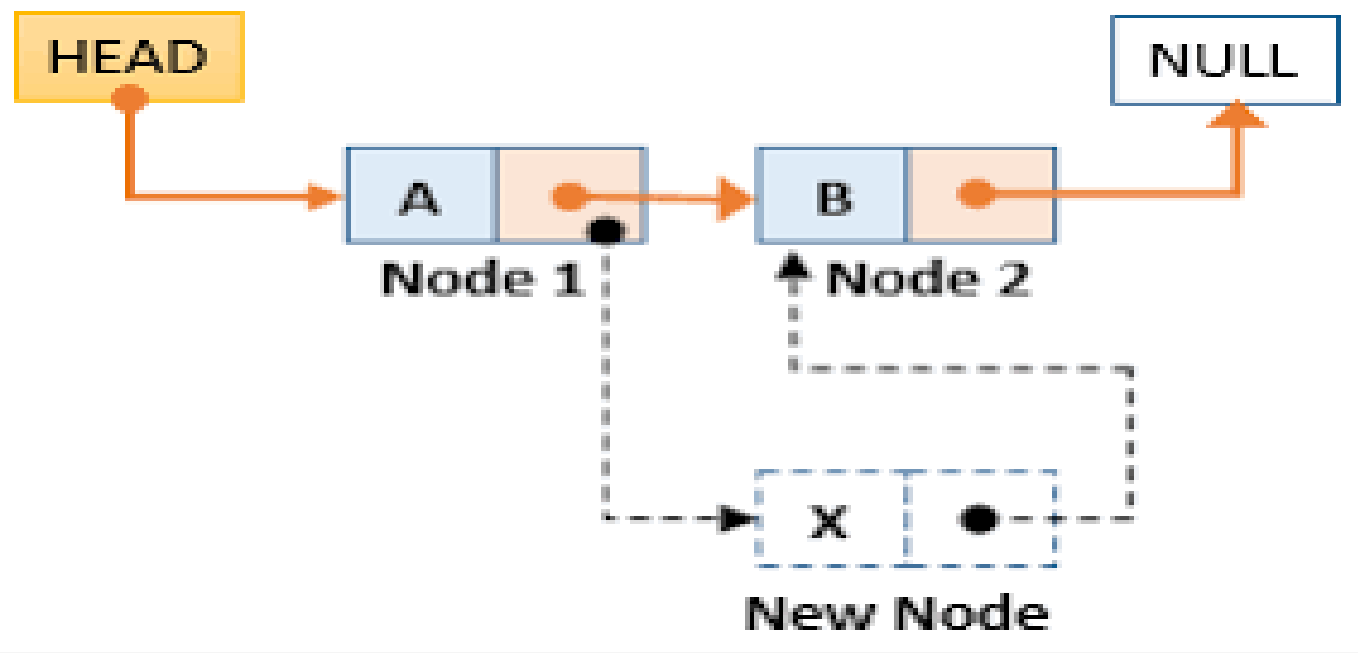
Autor:

LinkedList

Klasa LinkedList to dynamiczna struktura listy dwukierunkowej(każdy element przechowuj referencje do elementu następnego i elementu poprzedniego)



LinkedList – dodawanie elementu



ArrayList vs LinkedList

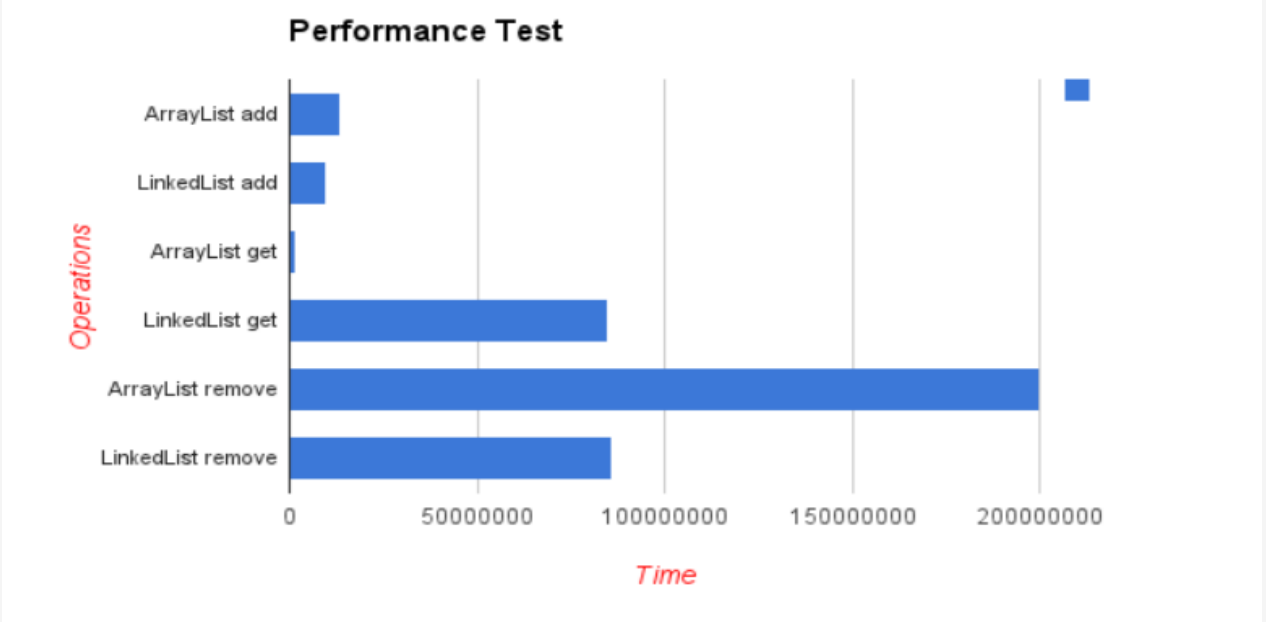
Porównanie złożoności niektóych operacji

	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1) amortized
remove()	O(n)	O(n)

Autor:

ArrayList vs LinkedList

Test wydajnościowy(czas w nano sekundach = 10^9 sekundy)

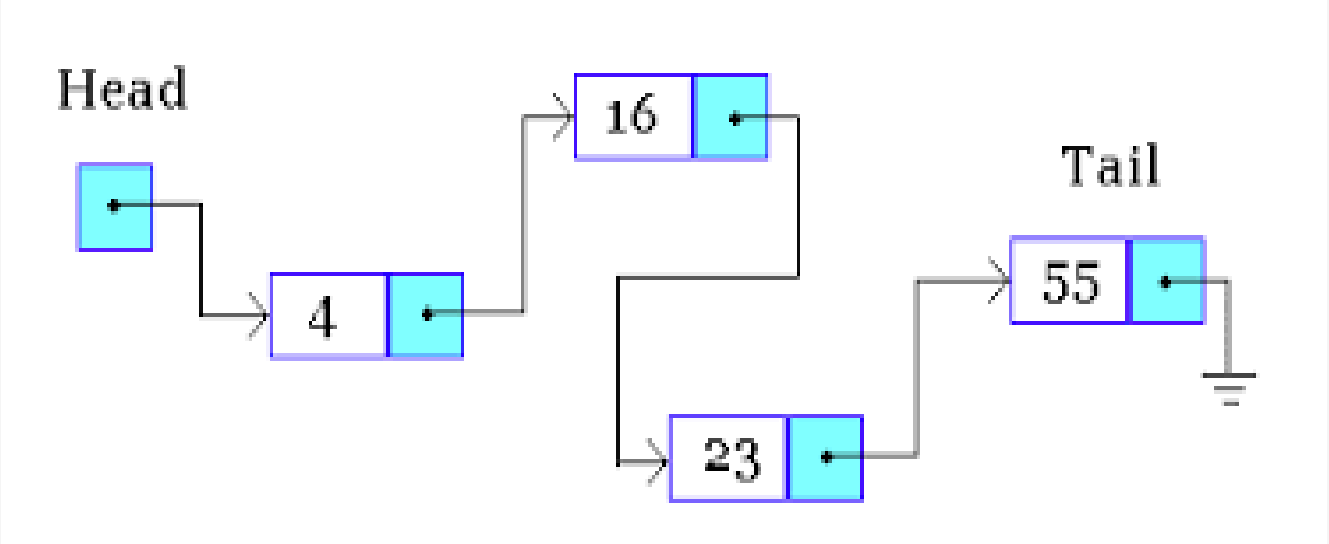


Zródło : http://www.programcreek.com/2013/03/arraylist-vs-linkedlist-vs-vector/

LinedList – Zadanie 1 – lista jednokierunkowa

Przejdz do klasy **MySingleLinedList.java**

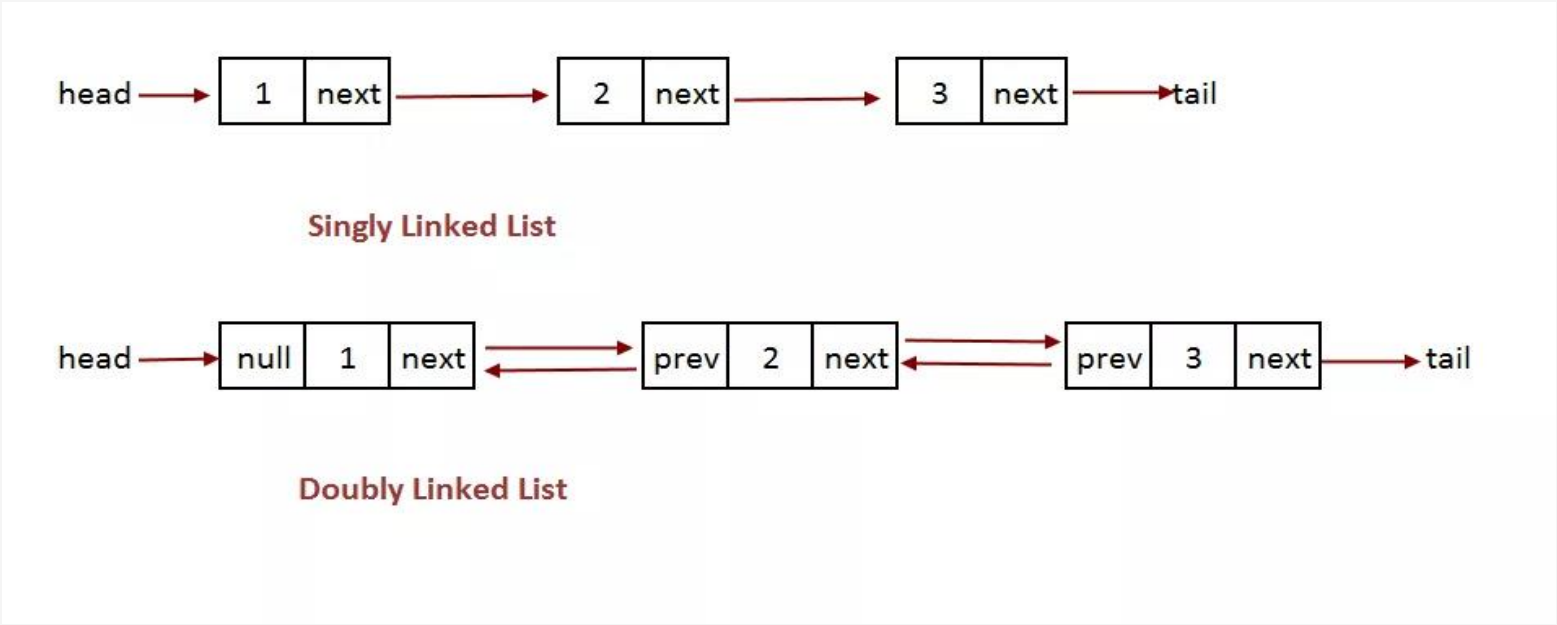
W klasie **MySingleLinedListTest.java** wykonaj operacje doprowadzając do przyjścia testów jednostkowych - uzupełniając implementację MyArrayList.java



LinedList – Zadanie 2

Przejdz do klasy **MyLinkedList.java**

W klasie **MyLinkedListTest.java** wykonaj operacje doprowadzając do przyjścia testów jednostkowych - uzupełniając implementację MyArrayList.java



Stos

Stos(ang. Stack) - jest liniową listą, w której elementy są dodwana i usuwane na samym końcu. Doskonałym przykładem jest "stos talerzy", umieszczony na stole, jeden na drugim. Kiedy potrzeba talerza, ten zdejmowany jest z wierzchołka stosu. Kiedy brudny talerz zostanie umyty, zostaje dodany na wierzołek stosu.

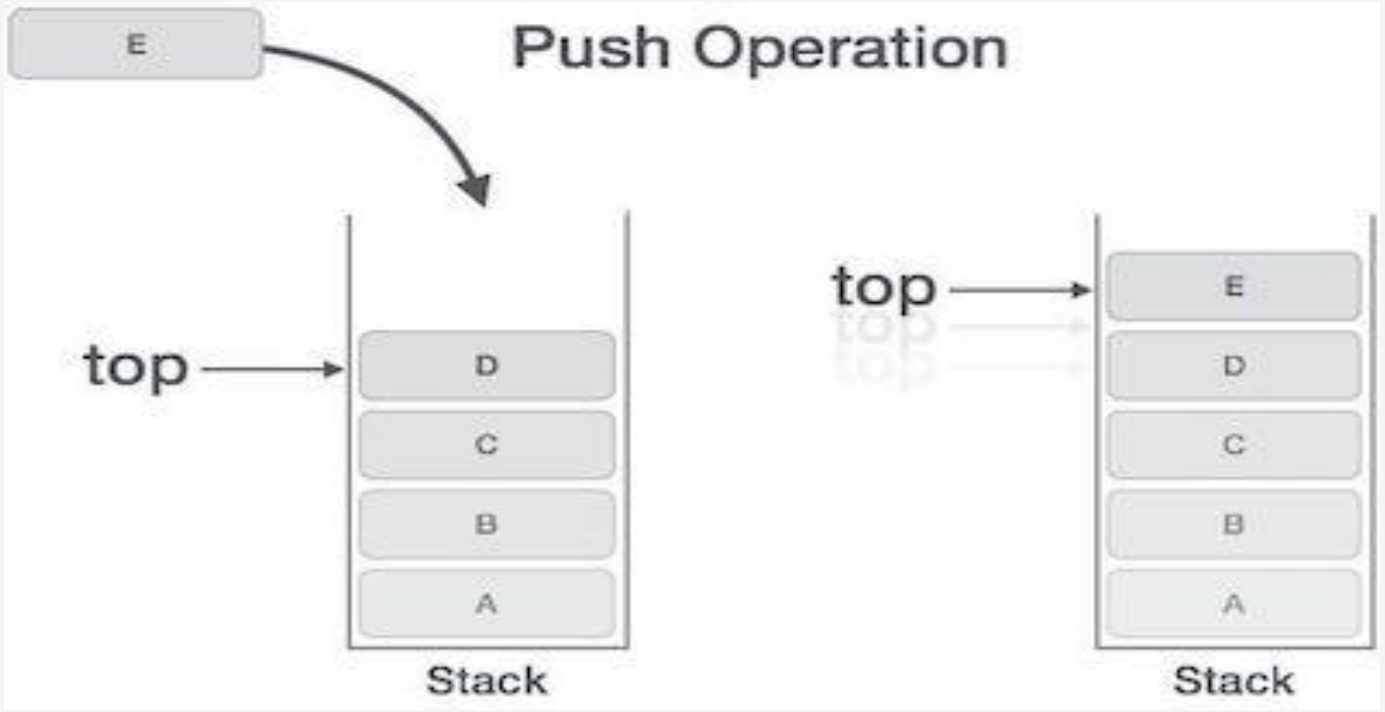
Stos

Stos to bufor typu **LIFO**, *Last In, First Out*; *ostatni na wejściu, pierwszy na wyjściu.*

Operację, jakie należy zdefiniować to:

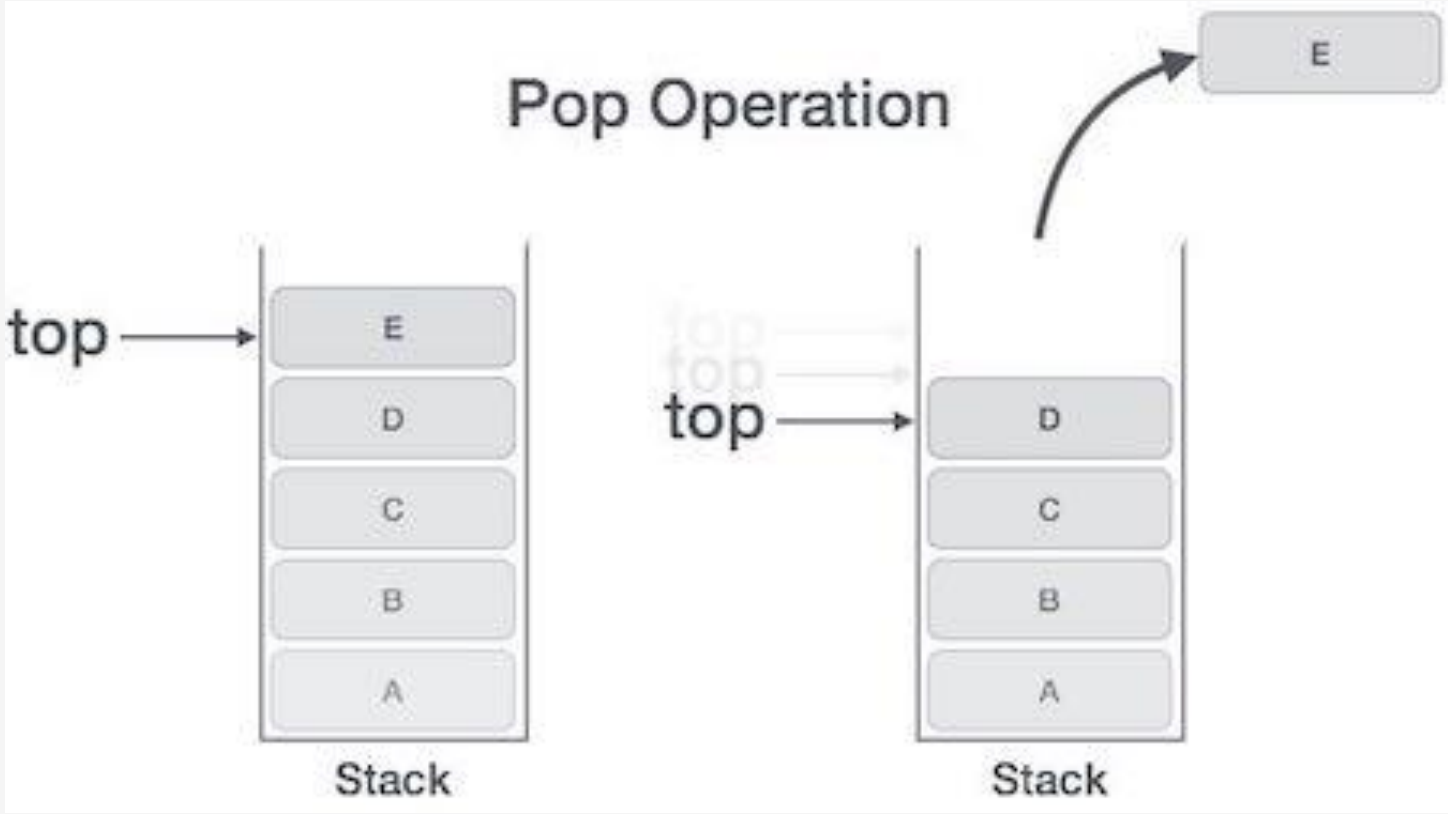
- void push(T data) -dodanie na stosie elementu.
- T pop() - pobranie elementu i usunięcie elementu ze stosu.

Stos – operacja push



Zródło: https://www.tutorialspoint.com/data_structures_algorithms/images/stack_push_operation.jpg

Stos – operacja pop



Zródło: https://www.tutorialspoint.com/data_structures_algorithms/images/stack_pop_operation.jpg

Stos – Zadanie domowe 1

Konwersja liczb dziesiątkowych na dwójkowe

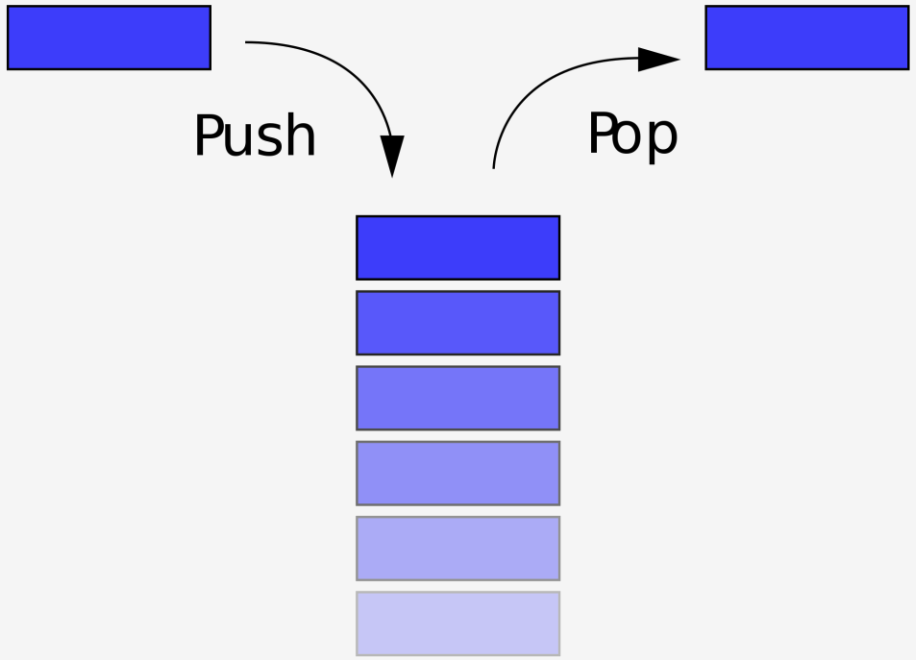
```
zainicjalizuj pusty stos S
wczytaj liczbę, n
while (n > 0)
    umieść n % 2 na stosie S
    n = n / 2
endwhile
while (S nie jest pusty) print pop(S)
```

https://eduinf.waw.pl/inf/alg/006_bin/0010.php

Stos – Zadanie

Przejdz do klasy **MyStack.java**

W klasie **MyStackTest.java** wykonaj operacje doprowadzając do przyjścia testów jednostkowych - uzupełniając implementację MyArrayList.java



Źródło: https://pl.wikipedia.org/wiki/Stos_(informatyka)#/media/File:Data_stack.svg

Kolejka

Kolejka(ang. Queue) - jest liniową listą, w której elementy są dodawane na jednym końcu, a usuwane na drugim. To nie znane kolejki "za komuny", ale życia codziennego można posłużyć się przykładem kolejki w której dołączamy na końcu a wychodzimy z niej na początku.

Kolejka – zastosowanie w informatyce

Jeśli należy wykonać kilka zadań, będą one umieszczone w kolejce. Przykładowo kilka osób chce coś wydrukować na drukarce sieciwowej. Ponieważ drukarka może wykonać tylko jedno zadanie w danej chwili, wszystkie pozostałe zostaną umieszczone w kolejce

Autor:

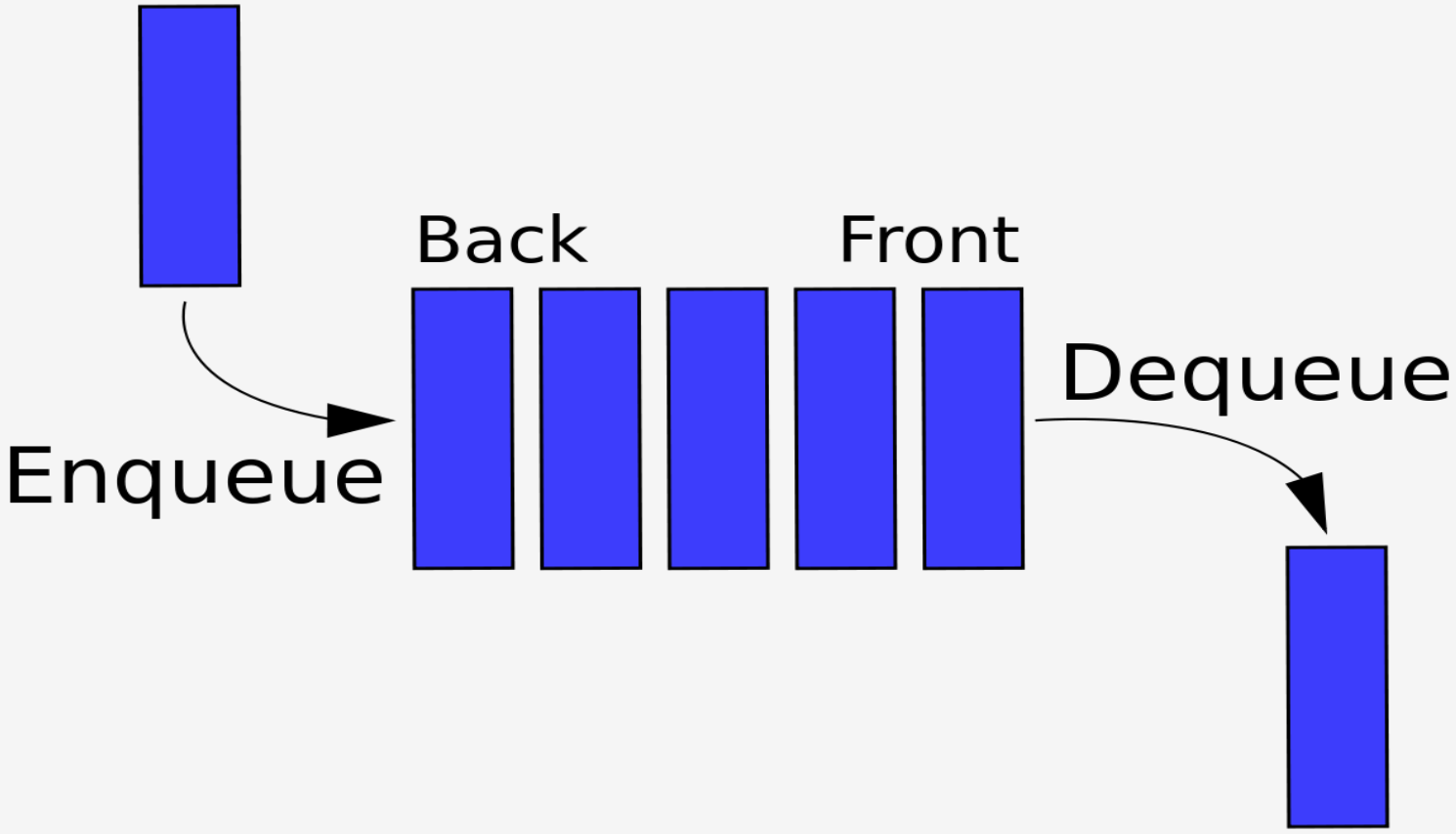
Kolejka – zastosowanie w informatyce

Kolejka to struktura **FIFO**(*FIRST IN , FIRST OUT*)

Lista operacja na kolejkach:

- Dodawanie elementu do kolejki(**enqueue**),
- Pobranie i usunięcie elementu z kolejki(**dequeue**)

Kolejka

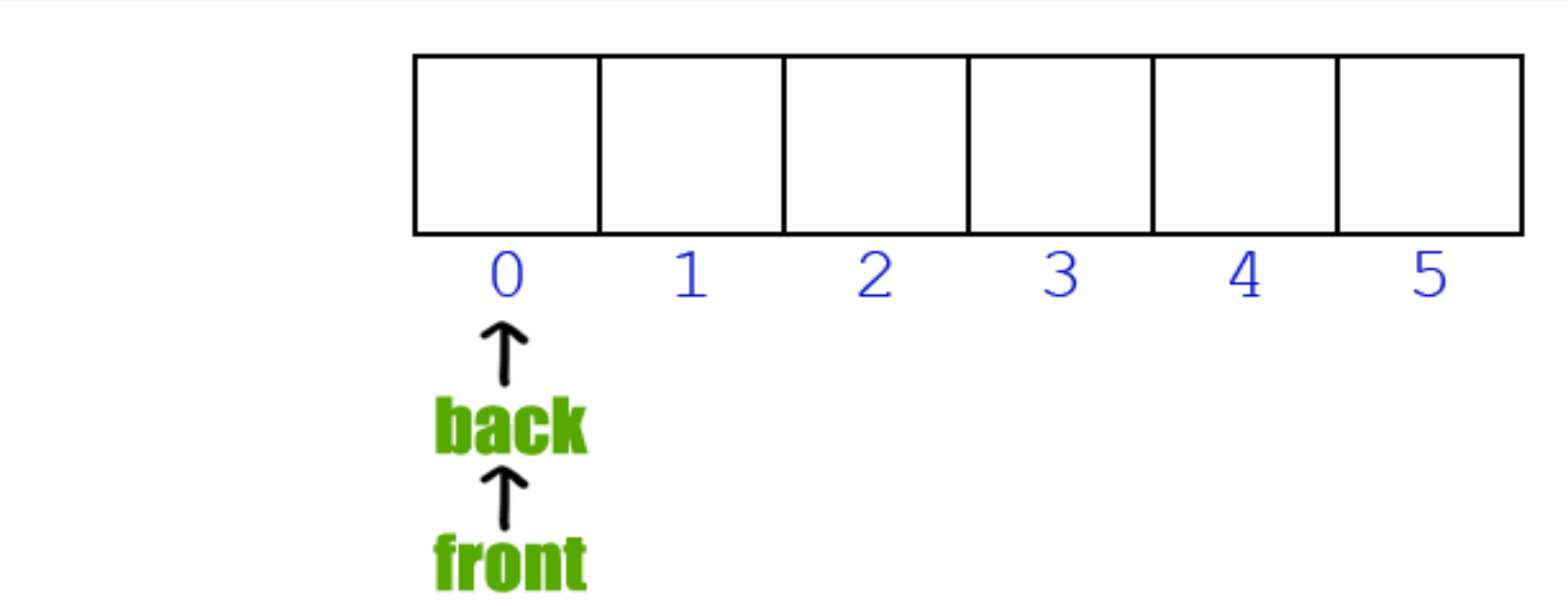


Źródło https://upload.wikimedia.org/wikipedia/commons/thumb/5/52/Data_Queue.svg/300px-Data_Queue.svg.png

Kolejki – Zadanie

Przejdz do klasy **MyQueue.java**

W klasie **MyQueueTest.java** wykonaj operacje doprowadzając do przyjścia testów jednostkowych - uzupełniając implementację MyArrayList.java



Źródło: https://i.gifer.com/R70N.gif

Złożoność obliczeniowa

- Analiza złożoności jest miarą pozwalającą niezależną od zmiennych czynników (procesor/ram itd.) pozwalającą określić wydajnośc danego algorytmu
- Miara która pozwala na dopasować do zdefiniowanej klasy algorytmów

Złożoność obliczeniowa

Większość prostych algorytmów należy do zaledwie kategorii:

- **Stałoczasowy** - algorytm jest stałocasowy, jeśli czas wykonania nie zależy od rozmiaru danych danych wejściowych (np. Pobranie i-tego elementu z tablicy)
- **Liniowy –** algorytm jest liniowy, jeśli czas jego wykonania jest proporcjonalny do rozmiaru danych wejściowych. Jeśli np. Wykonujemy sumowanie wszystkich elementów tablicy, to musimy odwołać się do n elemenetów
- **Kwadratowy –** algorytm jest kwadratowy, jeśli czas jego wykonania jest proporcjonalny do n_2. Jeśli np. Musimy porównywać każdy element listy ze wszystkimi innymi (*zagnieżdzone pętle for*)

Złożoność obliczeniowa

Gdy mówimy o wielkości problemu obliczeniowego, musimy zachować ostrożność w kwestii tego, jaką wielkość lub wielkości mamy na myśli.

Kuszącą drogę na skróty polegającą na liczeniu pętli.

- Jeśli mamy do czynienia z jedną pętlą, algorytm jest zwykle liniowy.
- Jeśli są dwie pętle (i jedna zagnieżdżona jest w drugiej), algorytm jest zwykle kwadratowy itd...

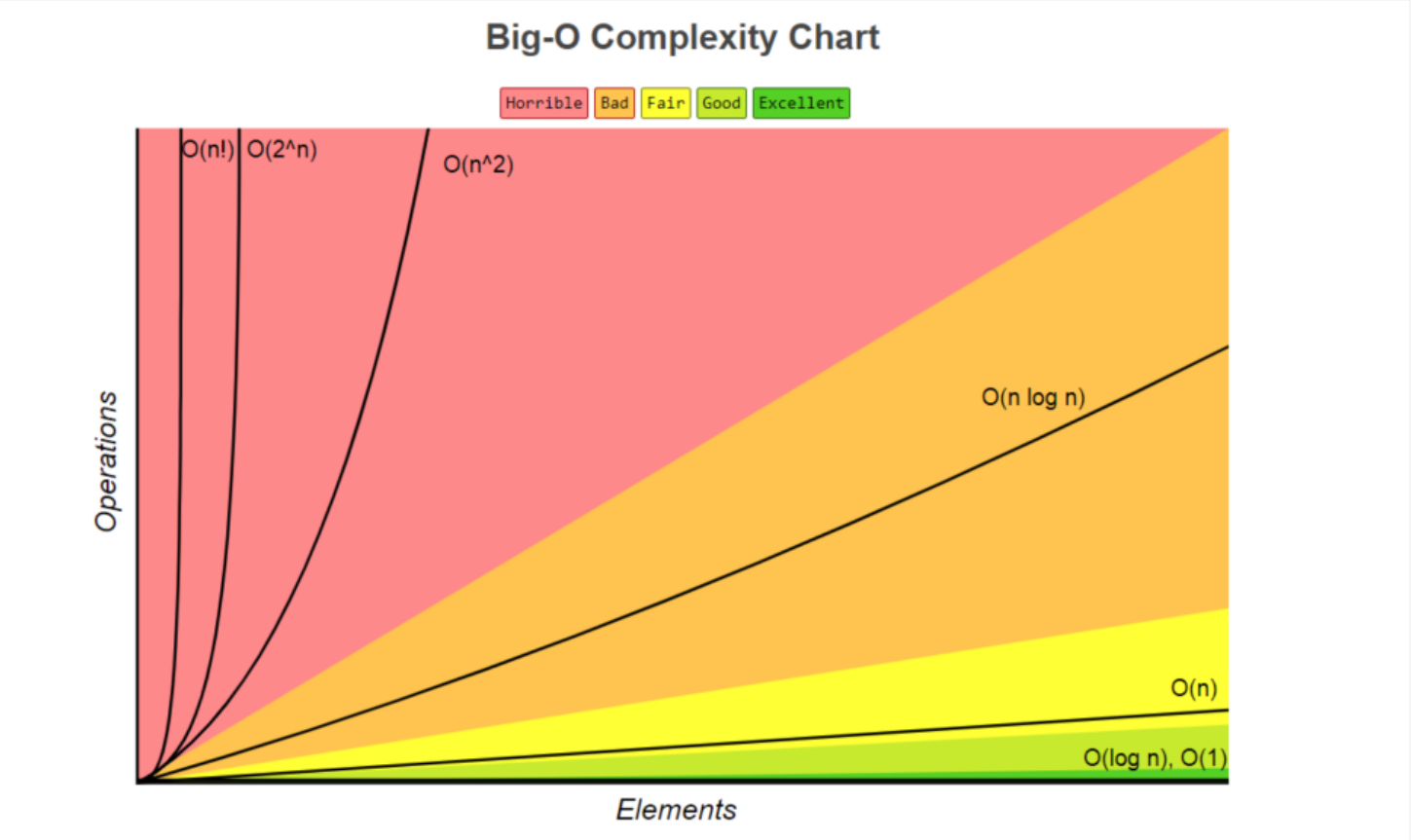
Zachowaj jednak czujność!

Powinieneś myśleć o tym, ile razy wykonuje się każda pętla.

Złożoność obliczeniowa

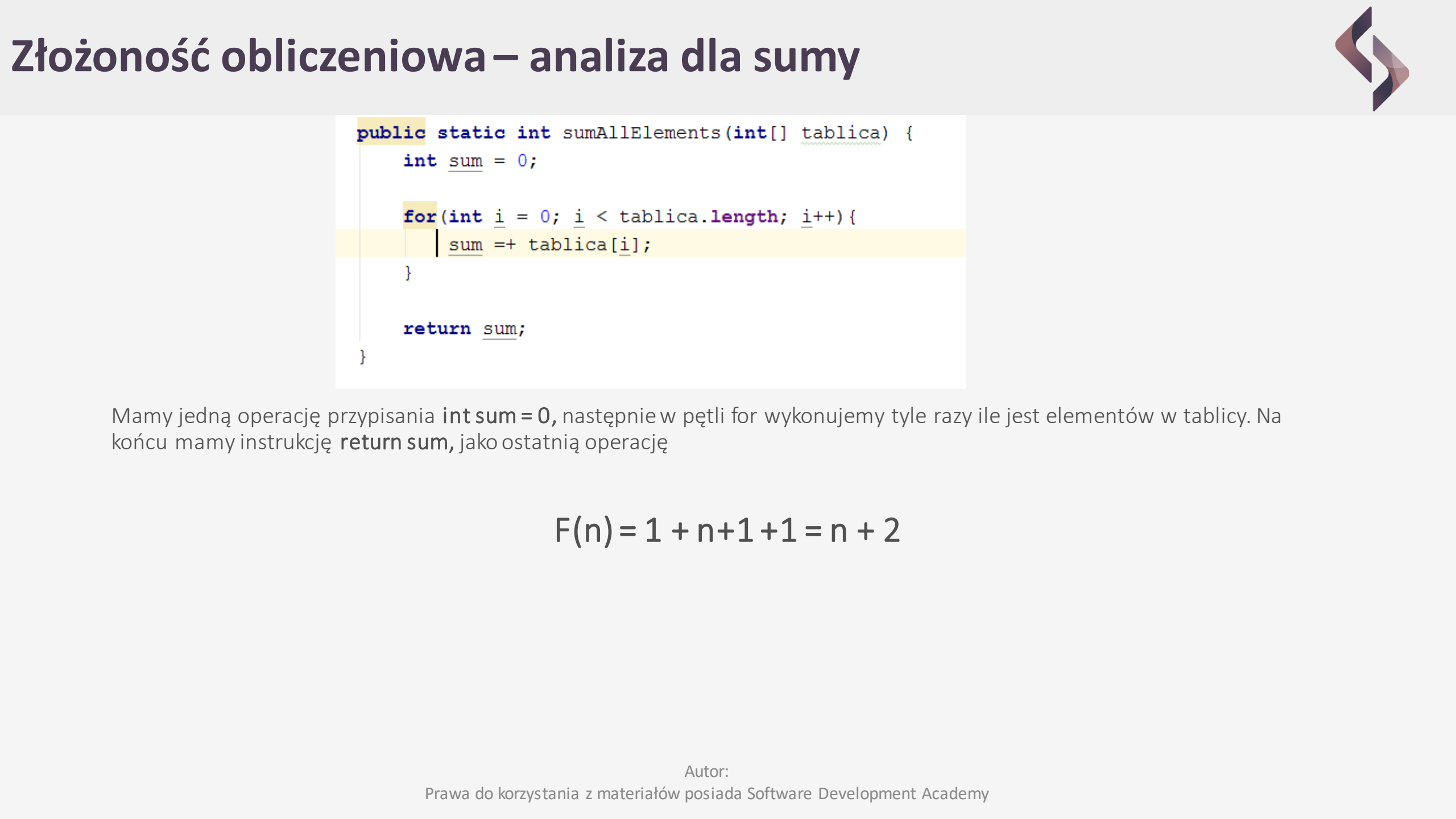
https://adrianmejia.com/blog/2014/02/13/algorithms-for-dummies-part-1-sorting/

Złożoność obliczeniowa



Źródło: http://bigocheatsheet.com/

Złożoność obliczeniowa – analiza dla sumy



```java
public static int sumAllElements(int[] tablica) {
    int sum = 0;

    for(int i = 0; i < tablica.length; i++){
        sum =+ tablica[i];
    }

    return sum;
}
```

Mamy jedną operację przypisania **int sum = 0,** następnie w pętli for wykonujemy tyle razy ile jest elementów w tablicy. Na końcu mamy instrukcję **return sum,** jako ostatnią operację

$$F(n) = 1 + n+1 +1 = n + 2$$

Złożoność obliczeniowa – Notacja O (dużego O)

- Wszystkie algorytmy stałoczasowe należą do zbioru nazywanego **O(1).** Zamiast tego mówić, że algorytm jest stałoczasowy, możemy powiedzieć, że należy do **O(1)**.
- Podobnie wszystkie algorytmy liniowe należą do zbioru **O(n) itd…**

W przypadku O-notacji, możemy ignorować stałe liczbowe, a więc dla poprzedniego przykładu:

$$F(n) = 1 + n+1 +1 = n + 2$$

Złożonośc algorytmu wynosi O(n)

* Bardziej matematyczne ujęcie O-notacji https://pl.wikipedia.org/wiki/Asymptotyczne_tempo_wzrostu

Autor:

Sortowanie

Sortowaniem nazywamy proces, w wyniku którego zbiór wartości zostaje uporządkowany w kolejności rosnącej bądź malejącej. Sortowania używamy w celu:

- Uzyskania bardziej czytelnego wyniku
- Przyspieszenia niektórych operacji(np. scalenie dwóch posortowanych list, wyszukiwanie elementu)

Sortowanie sortowanie przez wybieranie

Załóżmy że dysponujemy następująca tablicą liczb o zmiennej **tablica(czyli w języku java: *int[] tablica*)**, naszym zadaniem jest posortowanie w kolejności rosnącej, z wykorzystaniem metody sortowania przez wybieranie:

Wartość:	**51**	**49**	**75**	**67**	**18**
Indeks:	**0**	**1**	**2**	**3**	**4**

Sortowanie przez wybieranie

Pierwszy przebieg:

- Znajdujemy najmniejszą liczbę od pozycji **0 do 4** -> najmniejszą jest 18
- Zamieniamy elementy na pozycji 0 i 4

Wartość:	18	49	75	67	51
Indeks:	0	1	2	3	4

Sortowanie przez wybieranie

Drugi przebieg:

- Znajdujemy najmniejszą liczbę od pozycji **1 do 4** -> najmniejszą jest **49**
- Zamieniamy elementy na pozycji 1 i 1 (a w zasadzie nie robimy nic:))

Wartość:	**18**	**49**	**75**	**67**	**51**
Indeks:	**0**	**1**	**2**	**3**	**4**

Sortowanie przez wybieranie

Trzeci przebieg:

- Znajdujemy najmniejszą liczbę od pozycji **2 do 4** -> najmniejszą jest **51**
- Zamieniamy elementy na pozycji 2 i 4 (a w zasadzie nie robimy nic:))

Wartość:	**18**	**49**	**51**	**67**	**75**
Indeks:	**0**	**1**	**2**	**3**	**4**

Sortowanie przez wybieranie

Trzeci przebieg:

- Znajdujemy najmniejszą liczbę od pozycji **3 do 4** -> najmniejszą jest 67
- Zamieniamy elementy na pozycji 3 i 3 (a w zasadzie nie robimy nic:))

Wartość:	18	49	51	67	75
Indeks:	0	1	2	3	4

Sortowanie przez wybieranie

Wyniku wykonania algorytmu na **n-1** elementach , uzyskujemy posortowaną tablice:

Wartość:	**18**	**49**	**51**	**67**	**75**
Indeks:	**0**	**1**	**2**	**3**	**4**

Sortowanie przez wybieranie pseudo kod

Ogólnie, dla tablicy o wielkości *n*, należy wykonać *n-1* przebiegów. Pseudo kod dla sortowania przez wybieranie wygląda następująco:

For i = 0 to n - 2

min_pozycja**= pozycja najmniejszej wartości z zakresu od **tablica[i]** do **tablica[n-1]

*Zamień **tablica[i]** z **tablica[min_pozycja]***

Sortowanie przez wybieranie ZADANIE

- Na podstawie pseudokodu, w klasie **SelectionSort.java**
- Należy zaimplementować algorytm sortowania przez wybieranie.
- W celu weryfikacji poprawności - sprawdź wykonanie testu **SelectionSortTest.java**

For i = 0 to n - 2

__min_pozycja__= pozycja najmniejszej wartości z zakresu od __tablica[i]__ do __tablica[n-1]__

Zamień __tablica[i]__ z __tablica[min_pozycja]__

Sortowanie przez wybieranie - Analiza

- W celu znalezienia najmniejszego elementu spośród n elementów , wykonujemy n – 1 porównań(*`getIndexOfNextMinElement`*). W pierwszym przebiegu wykonujemy n-1 porównań , by znaleźć najmniejszy spośród n elementów, następne n-2 i tak dalej aż do ostatniego elementu w którym wykonujemy jedno porównanie.

- Dodatkowo obliczenia wykonujemy **n-1** razy.

Złożoność **(n-1)(n-1) = n^2 –1 = n^2 => O(n^2)**

Sortowanie przez wstawianie

Wyobraź sobie, że mamy karteczki z liczbami jak tablicy poniżej.

- Podnosimy w kolejność, a więc najpierw mamy karteczkę 52, a następnie 41 itd.
- Kiedy podnosimy każdą następną karteczkę dodajemy je do trzymanych kart tak, żeby był posortowane
- Kiedy podnosimy 52 to będziemy mieli tylko jedną karteczkę z liczbą - uznajemy że jest posortowana
- Następnie podnosimy liczbę **41** i umieścimy ją przed **52** czyli mamy **(41,52)**
- Podnosimy **79** umieścimy ją za liczbą **52,** czyli mamy **(41,52,79)**
- Podnosimy **17** i umieszczamy ją przed **41**, czyli mamy **(17,41,52,79)**
- Podnosimy **32** i umieszczamy ją za **17 ,** czyli mamy **(17,32,41,52,79)**

Wartość:	**52**	**41**	**79**	**17**	**32**
Indeks:	**0**	**1**	**2**	**3**	**4**

Sortowanie przez wstawianie

Stan przed sortowaniem:

Wartość:	52	41	79	17	32
Indeks:	0	1	2	3	4

Pierwszy przebieg:

- Przetwarzamy pierwszą liczbę czyli 41 (Indeks pierwszy możemy pominąć). Operacja sprowadza się do umieszczenia przetwarzanej liczby w taki sposób, że dwie pierwsze liczby są posortowane.

Wartość:	41	52

Sortowanie przez wstawianie

Stan przed sortowaniem:

Wartość:	52	41	79	17	32
Indeks:	0	1	2	3	4

Drugi przebieg:

- Przetwarzamy **tablica[2]** czyli liczbę **79**. Operacja to sprawdza się do umieszczenia przetwarzanej liczby w taki sposób, że trzy pierwsze liczby zapisane w tablicy będą posortowane.

Wartość:	41	52	79

Sortowanie przez wstawianie

Stan przed sortowaniem:

Wartość:	52	41	79	17	32
Indeks:	0	1	2	3	4

Trzeci przebieg:

- Przetwarzamy **tablica[3]** czyli liczbę 17. Operacja to sprawdza się do umieszczenia przetwarzanej liczby w taki sposób, że trzy pierwsze liczby zapisane w tablicy będą posortowane.

Wartość:	17	41	52	79

Sortowanie przez wstawianie

Stan przed sortowaniem:

Wartość:	52	41	79	17	32
Indeks:	0	1	2	3	4

Trzeci przebieg:

- Przetwarzamy **tablica[4]** czyli liczbę 32. Operacja to sprawdza się do umieszczenia przetwarzanej liczby w taki sposób, że trzy pierwsze liczby zapisane w tablicy będą posortowane.

Wartość:	17	32	41	52	79

Sortowanie przez wstawianie ZADANIE

- Na podstawie pseudokodu, w klasie `InsertionSort.java`
- Należy zaimplementować algorytm sortowania przez wybieranie.
- W celu weryfikacji poprawności - sprawdź wykonanie testu `InsertionSortTest.java`

```
0. Insert_sort(A, n)

1.  for i=2 to n :
2.      klucz = A[i]
3       # Wstaw A[i] w posortowany ciąg A[1 ... i-1]
4.      j = i - 1

5.      while j>0 and A[j]>klucz:
6.          A[j + 1] = A[j]
7.          j = j - 1
8.      A[j + 1] = klucz
```

ZADANIE DOMOWE 2

W klasach BubbleSort oraz BubbleSortTest(test jednostkowy) zaimplementuj algorytm:

- **sortowania bąbelkowego (**http://www.algorytm.org/algorytmy-sortowania/sortowanie-babelkowe-bubblesort/bubble-2-j.html)

ZADANIE DOMOWE 3

Zapoznaj się z algorytmem do sprawdzenia czy dwa tablice znaków *char[] ch1 i char[] ch2* są anagramem.

Metoda **public boolean isAnagram(char[] ch1, char[] ch2)**

Uwaga ! Nie korzystaj z klasy String.

https://javaconceptoftheday.com/anagram-program-in-java/

Na następnych zajęciach

W zajęciach następnych omówimy następujące zagadnienia:

- Struktury danych
  - Listy podwieszane
  - Listy cykliczne
  - Zbiory
  - Kolejka priorytetowa
- Rekurencja
- Algorytmy:
  - Haszowanie
  - Wyszukiwanie binarne
  - Quick sort
  - Merge sort
  - Heapsort
  - Wprowadzenie do zagadnień drzew binarnych

Autor:

Przydatne strony

https://eduinf.waw.pl/inf/alg/001_search/0086.php - o algorytmach i strukturach danych

http://www.algorytm.org/ - duży zbiór gotowych implementacji prostych algorytmów

http://www.hackerearth.com/practice - świetna strona , zawiera opis poszczególnych algorytmów, w języku angielskim