Programowanie 1

Mateusz Duraj

Agenda

- Zestaw narzędzi potrzebny na zajęciach
- Algorytmy na przykładzie wyszukiwania elementów i sortowania
- Przykładowe algorytmy zadawane podczas rekrutacji
- Złożoność obliczeniowa
- Struktury danych na przykładzie stosu i kolejki
- Wprowadzenie do Java Collection Framework
- Mapy i haszowania

Algorytm trudne słowo , na proste rzeczy

Autor:

Algorytm – zestaw narzędzi potrzebnych w Javie

Instrukcja switch

```
Color color = ...
switch (color) {
case GREEN:
    /* 1 */
    break;
case RED:
    /* 2 */
    break;
}
```

Algorytm – zestaw narzędzi potrzebnych w Javie

Operatory warunkowe

```
int number = ...;
if (number % 2 == 0) {
    System.out.println("Number is even!");
} else {
    System.out.println("Number is odd!");
}
```

Algorytm – zestaw narzędzi potrzebnych w Javie

Pętle for, while, do-while

```
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 5; ++j) {
        System.out.println(j);
    }
}
```

Autor:

Algorytm – zestaw narzędzi potrzebnych w Javie

Konstrukcje złożone

```
int number = 0;
while (number < 10) {
    System.out.println(number);
    if (number < 5) {
        ++number;
        continue;
    }
    number += 2;
}
```

Złożoność obliczeniowa

- Analiza złożoności jest miarą pozwalającą niezależną od zmiennych czynników (procesor/ram itd.) pozwalającą określić wydajnośc danego algorytmu

Złożoność obliczeniowa

Wyróżniamy dwa rodzaje złozoności:

- **Czasową** - opisuje ilość czasu potrzebnego do wykonania algorytmu
- **Pamięciową** - jest miarą opisującą wykorzystanie pamięci

Złożoność obliczeniowa, n – liczba elementów

- **Stałoczasowy O(1)** - algorytm jest stałocasowy, jeśli czas wykonania nie zależy od rozmiaru danych danych wejściowych (np. Pobranie i-tego elementu z tablicy)
- **Logarytmiczny O(logn) -** klasa logarytmiczna , gdy podwajanie liczby elementów, nie podwaja czasu(wyszukiwanie binarne)
- **Liniowy O(n)–** algorytm jest liniowy, jeśli czas jego wykonania jest proporcjonalny do rozmiaru danych wejściowych(iterowanie po wszystkich elementach tablicy)
- **Quasi(ang. Prawie) Liniowy O(n logn) -** Każdy element tablicy musi być porównany z pozostałymi - bardzo dużo operacji porównania(Algorytmy sortowania)
- **Kwadratowy O(n^2)–** 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*)

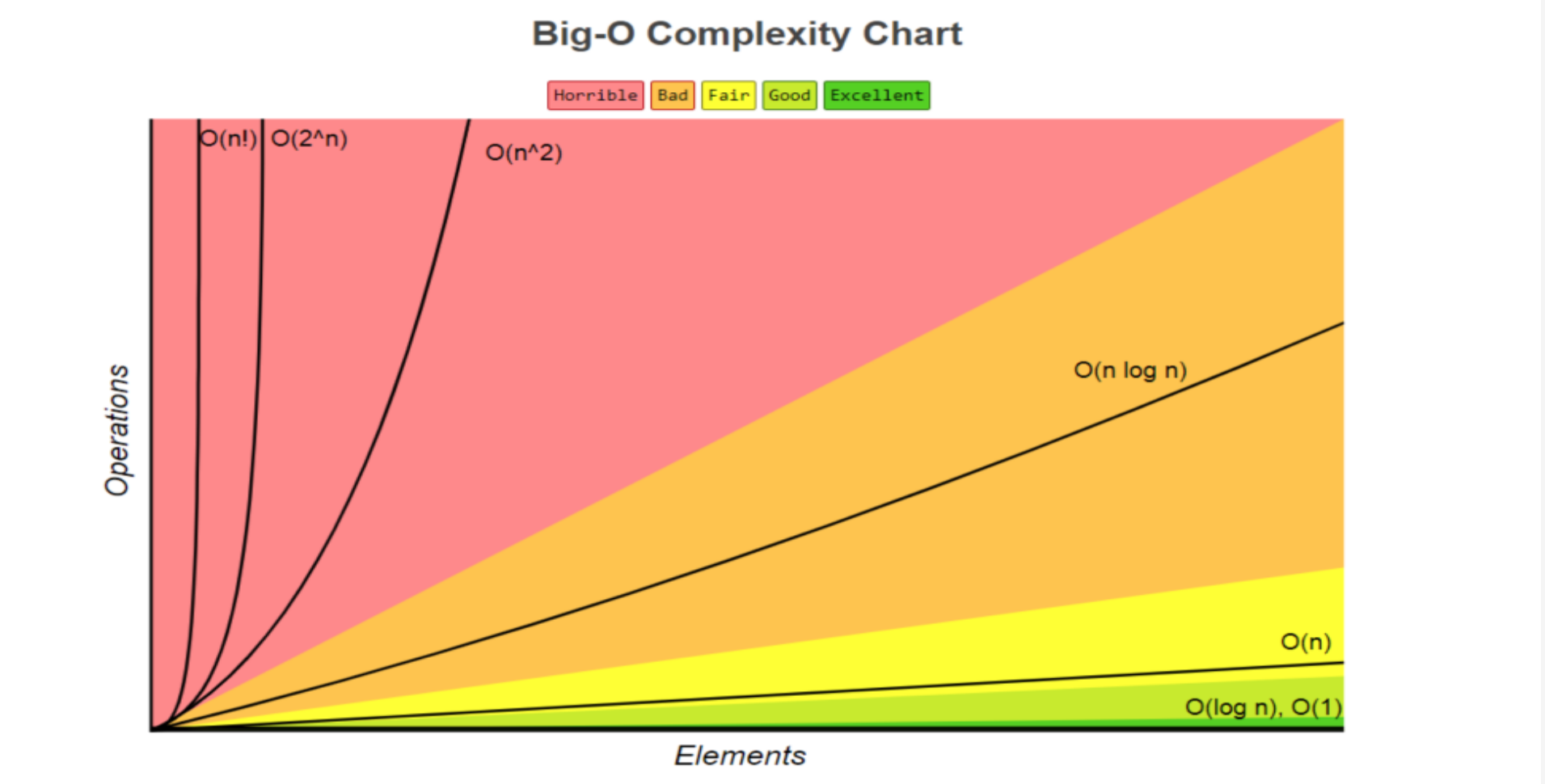
Autor:

Złożoność obliczeniowa – klasy złożoności

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
logarytmiczna (log n)	0	0	0	1	1	1	1	1	1	1	1	1	1	1
liniowa (n)	1	2	3	4	5	6	7	8	9	10	11	12	13	14
liniowo logarytmiczna (nlog n)	0	1	1	2	3	5	6	7	9	10	11	13	14	16
kwadratowa (n^2)	1	4	9	16	25	36	49	64	81	100	121	144	169	196
sześcienna (n^3)	1	8	27	64	125	216	343	512	729	1000	1331	1728	2197	2744
wykładnicza (2^n)	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
silnia (n!)	1	2	6	24	120	720	5040	40320	362880	3628800	39916800	4,79E+08	6,23E+09	8,72E+10

Autor:

Złożoność obliczeniowa– klasy złożoności



Źródło: http://bigocheatsheet.com/

Złożoność czasowa– 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

$$F(n) = ?$$

Złożoność obliczeniowa – Notacja O (dużego O)

W przypadku O-notacji, możemy ignorować stałe liczbowe:

- $F(n) = 1 + n+1 +1 = n + 2$
- $F(n) = 0.5 * n = n$

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

Autor:

Złożoność pamięciowa– Notacja O (dużego O)

Podobnie jak złożoność czasowa jest miarą czasu działania algorytmu, tak złożoność pamięciowa jest miarą ilości wykorzystanej pamięci.

Autor:

Złożoność obliczeniowa

- Otwórz klasę **BigO.java** -przykłady złożoności

Złożoność obliczeniowa – co potrzebujemy do rozmowy o pracę

- Znać złożoności podstawowych operacji na strukturach danych (*add/remove/edit/update*) dla List, Kolejek,Stosu,Map
- Złożoność obliczeniową podstawowych algorytmów sortowania
- Okreslić złozonośc przestawionych prostych algorytmów (jak np. w BigO.java)

Autor:

Tablica (ang. Array) - statyczna struktura danych

1. Do konkretnego elementu odwołujemy się za pomocą klucza (indeksu)
2. Stały rozmiar deklarowany przy tworzeniu tablicy
3. Gwarantowany dostęp w tym samym czasie do każdego elementu tablicy

```
// deklaracja
int[] tablica;
// inicjalizacja n elementowej tablicy
tablica = new int[n];
// przypisanie wartości 13 n-temu elementowi tablicy
tablica[n] = 1;
// pobranie długości tablicy
int dlugosc = tablica.length;
// kopiowanie tablicy
int[] kopia = tablica.clone();
```

Tablica zadanie - klasa `TableTask1`

1. Wypełniamy tablicę kolejnymi wartościami od 1 do 10
   czyli: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2. Co drugą liczbę zwiększamy o wartość jej poprzednika(pierwsza pętla)
   oczekujemy: [0, 1, 2, 5, 4, 9, 6, 13, 8, 17]
3. Każdą liczbę parzystą(nie indeks) dzielimy przez 2 (druga pętla)
   oczekujemy: [0, 1, 1, 5, 2, 9, 3, 13, 4, 17]
4. Sumujemy wszystkie liczby w tablicy
   oczekujemy: 55

Sortowanie na przykładzie prostych algorytmów

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)

Autor:

Sortowanie bąbelkowe (ang. Bubble sort)

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności

$$[\underbrace{4,2}_{4>2},5,1,7] \rightarrow [2,\underbrace{4,5}_{4<5},1,7] \rightarrow [2,4,\underbrace{5,1}_{5>1},7] \rightarrow [2,4,1,\underbrace{5,7}_{5<7}]$$

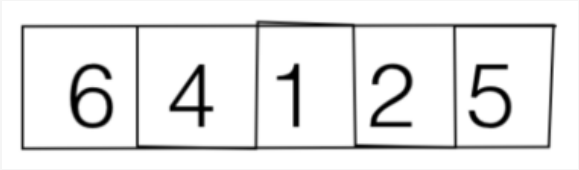
$$[\underbrace{2,4}_{2<4},1,5,7] \rightarrow [2,\underbrace{4,1}_{4>1},5,7] \rightarrow [2,1,\underbrace{4,5}_{4<5},7]$$

$$[\underbrace{2,1}_{2>1},4,5,7] \rightarrow [1,\underbrace{2,4}_{2<4},5,7]$$

$$[\underbrace{1,2}_{1<2},4,5,7]$$

Sortowanie bąbelkowe (ang. Bubble sort)

Mamy następująca tablice:



Iteration 1:	Iteration 2:	Iteration 3:	Iteration 4:
6 4 1 2 5	**4 1** 2 5 6	**1 2** 4 5 6	**1 2** 4 5 6
4 **6 1** 2 5	1 **4 2** 5 6	1 **2 4** 5 6	1 2 4 5 6
4 1 **6 2** 5	1 2 **4 5** 6	1 2 4 5 6	
4 1 2 **6 5**	1 2 4 5 6		
4 1 2 5 6			

Sortowanie bąbelkowe ZADANIE

Zaimplementuj algorytm sortowania bąbelkowego , korzystając z poniższego pseudokodu w klasie **BubbleSort**. Pamiętaj o testach w BubbleSortTest.java

```
n = array.length
for i = 0 to n-1 do:

    for j = 0 to n-1 do:


        if list[j] > list[j+1] then
            //Zamień elementy tablicy
            swap( list[j], list[j+1] )
        end if

    end for

end for
```

Sortowanie bąbelkowe PYTANIE

Jaką złożoność obliczeniową ma algorytm sortowania bąbelkowego?

Autor:

Sortowanie przez wybieranie(ang. Selection sort)

Algorytm przedstawia się następująco:

- Wyszukaj minimalną wartośc z tablicy spośród elementów od i do końca tablicy
- Zamień wartość mimimalną, z elementem na pozycji i

Sortowanie przez wybieranie(ang. Selection sort)

Przykładowo dla tablicy: int array = new int[] {9,1,6,8,4,3,2,0}

nr iteracji (wartość i)	tablica	minimum
0	[9,1,6,8,4,3,2,0]	0
1	[0,1,6,8,4,3,2,9]	1 (element znajduje się na właściwej pozycji)
2	[0,1,6,8,4,3,2,9]	2
3	[0,1,2,8,4,3,6,9]	3
4	[0,1,2,3,4,8,6,9]	4 (element znajduje się na właściwej pozycji)
5	[0,1,2,3,4,8,6,9]	6
6	[0,1,2,3,4,6,8,9]	8 (element znajduje się na właściwej pozycji)

Autor:

Sortowanie przez wybieranie(ang. Selection sort)

- Na podstawie pseudokodu, w klasie **SelectionSort.java**
- Należy zaimplementować algorytm sortowania przez wybieranie.
- W celu weryfikacji poprawności - sprawdź wykonanie testu **SelectionSortTest.java**
- Określ złozoność algorytmu
- Czy da się jakoś przyspieczyć działanie*

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 wstawianie – ZADANIE DOMOWE

- 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
```

Algorytmy wyszukiwania liniowego

- Przeglądaj kolejne elementy zbioru. Jeśli przeglądany element posiada odpowiednie własności(np. jest liczbą o poszkiwanej wartości), to zwróć jego pozycję w zbiorze. Jeżeli nie ma odpowiedniego elementu zwróć -1.

Algorytmy wyszukiwania liniowego - Zadanie

- Zaimplementuj w klasie **LinearSearch** i przetestuj w **LinearSerachTest**
- Jaka jest złożoność algortmu?

Algorytmy wyszkiwania – wyszukiwanie binarne

- Wyszukiwanie binarne jest algorytmem opierającym się na ***metodzie dziel i zwyciężaj.***
- W czasie logarytmicznym algorytm stwierdza, czy szukany element znajduj się w **uporządkowanej** tablicy.

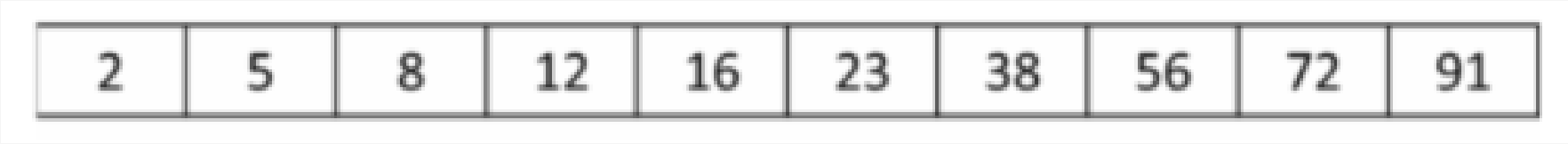
Wyszukiwanie binarne vs liniowe

- Wyszukiwanie liniowe – iterujemy po każdym elementcie tablicy.
- Dla tablicy elementów zawierającej milion elementów, wyszukiwanie binarne musi sprawdzić 20 elementów $(\log_2 1\,000\,000 \approx 20)$
- Dla porównania wyszukiwanie liniowe wymaga w najgorszym przypadku przejrzenia wszystkich elementów tablicy

Autor:

Wyszukiwanie binarne

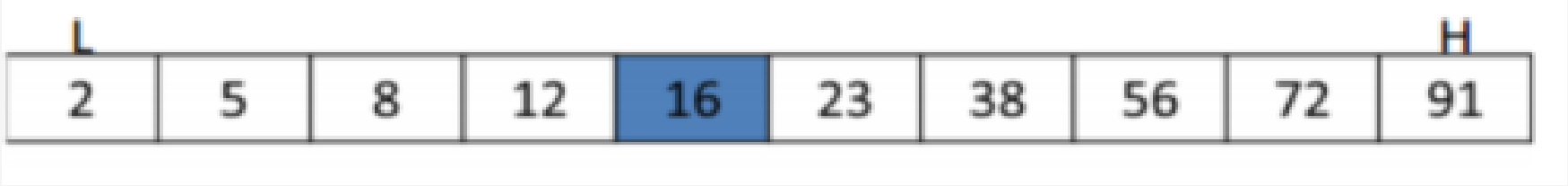
- Dana jest następująca tablica i szukamy liczby 23:



Wyszukiwanie binarne

Dana jest następująca tablica i szukamy liczby 23:

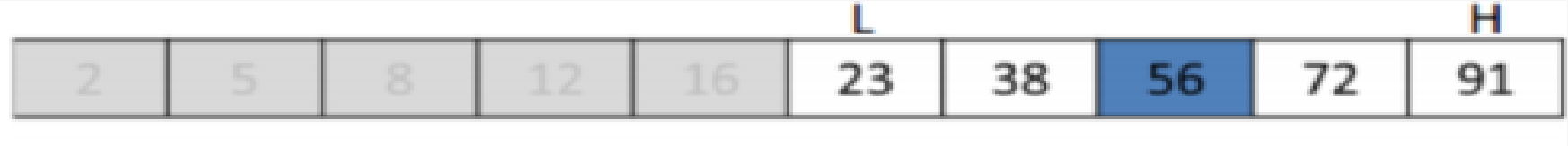
Piewsza iteracja array[middle]=16 < 23 , więc szukamy w 2 połowie tablicy:



Wyszukiwanie binarne

Dana jest następująca tablica i szukamy liczby **23**:

Druga iteracja array[middle]=56 < 23 , więc szukamy w 1 połowie tablicy:

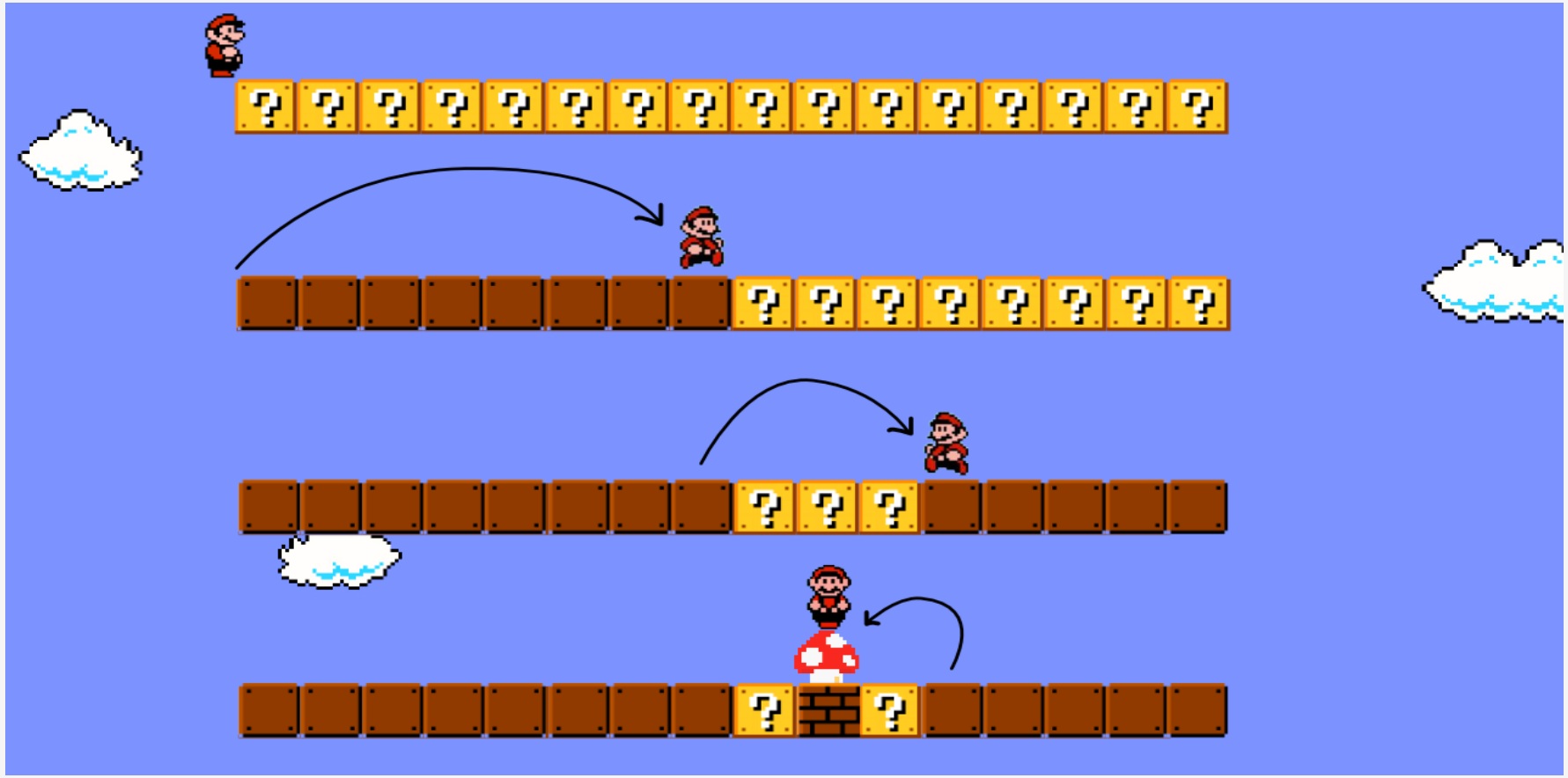


Wyszukiwanie binarne

Dana jest następująca tablica i szukamy liczby 23:

Trzecia iteracja array[middle]=23 = 23 , element znajduję się na 5 pozycji:

Wyszukiwanie binarne



https://www.topcoder.com/wp-content/uploads/2017/07/binary-search.png

Wyszukiwanie binarne – Zadanie

Dla poniższego pseudokodu, zaimplementuj w klasie **BinarySearchAlgorithm.java,** metodę ***binarySearch*** **,** oraz sprawdz jej poprawność wykorzystując testy **BinarySearchAlgorithmTest.java**

```
binarySearch(A[1..N], numberToSearch)
        min←0, max←N, min←0
        while(max != min)
                mid = (min + max)/2
                        if(A[mid] == search)
                                return mid
                        else if(a[mid]<search)
                                min = mid
                        else
                                max = mid
        return -1
```

ALGORYTMY - ZADANIA REKRUTACYJNE 1

Dokończ implementacje w klasach algorytmu:

- Anagram.java

ALGORYTMY - ZADANIA REKRUTACYJNE 2

Dokończ implementacje w klasach algorytmu:

- StringReversing.java

ALGORYTMY - ZADANIA REKRUTACYJNE 3

Dokończ implementacje w klasach algorytmu:

- IntReverser.java

ALGORYTMY - Na co się przygotować do rozmowy

- Zaimplementować prosty algorytm (wyszukiwanie elementu maksymalnego, anagram itp..)
- Znać proste algorytmy sortowania (bąbelkowe, przez wstawianie)
- Potrafić określić złozonośc obliczeniową podanego algorytmu

ALGORYTMY – GDZIE SZUKAĆ ZADAŃ

- https://www.hackerearth.com/practice/ - na początek wiedza teoretyczna + praktyczna
- https://pl.spoj.com/ - bardzo duża baza danych z zadaniami z algorytmów
- https://www.hackerrank.com/ - bardzo podobna platforma do spoja, firmy dość często weryfikują z użyciem tej platformy potencjalnych kandydatów

Wiecej o online platformach do rozwiązywania problemów z algorytmów w wątpu:

https://www.quora.com/Should-I-do-HackerRank-SPOJ-TopCoder-CodeForces-or-CodeChef

ALGORYTMY - ZADANIA DOMOWE

Zapoznaj się i zaimplementuj algorytmy:

- **Sita Eratostenesa** (https://pl.wikipedia.org/wiki/Sito_Eratostenesa)
- **Szyfru cezara**(https://pl.wikipedia.org/wiki/Szyfr_Cezara)

Struktury danych

Dotychczas omówiliśmy :

- Tablice (ang. Array) - statyczne struktury danych(musimy na początku podać rozmiar danych
- Listy jednokierunkowe i dwukierunkowe

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.

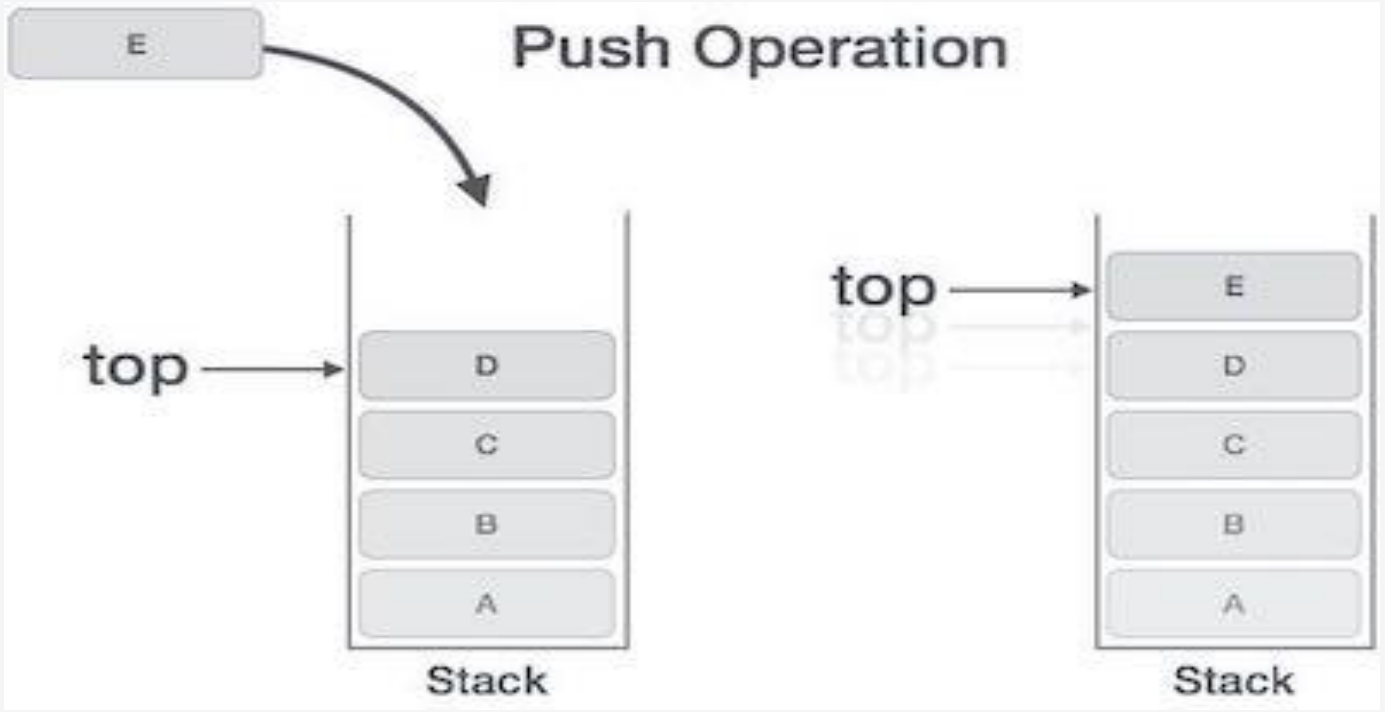
Stos

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

Metody dla Stosu:

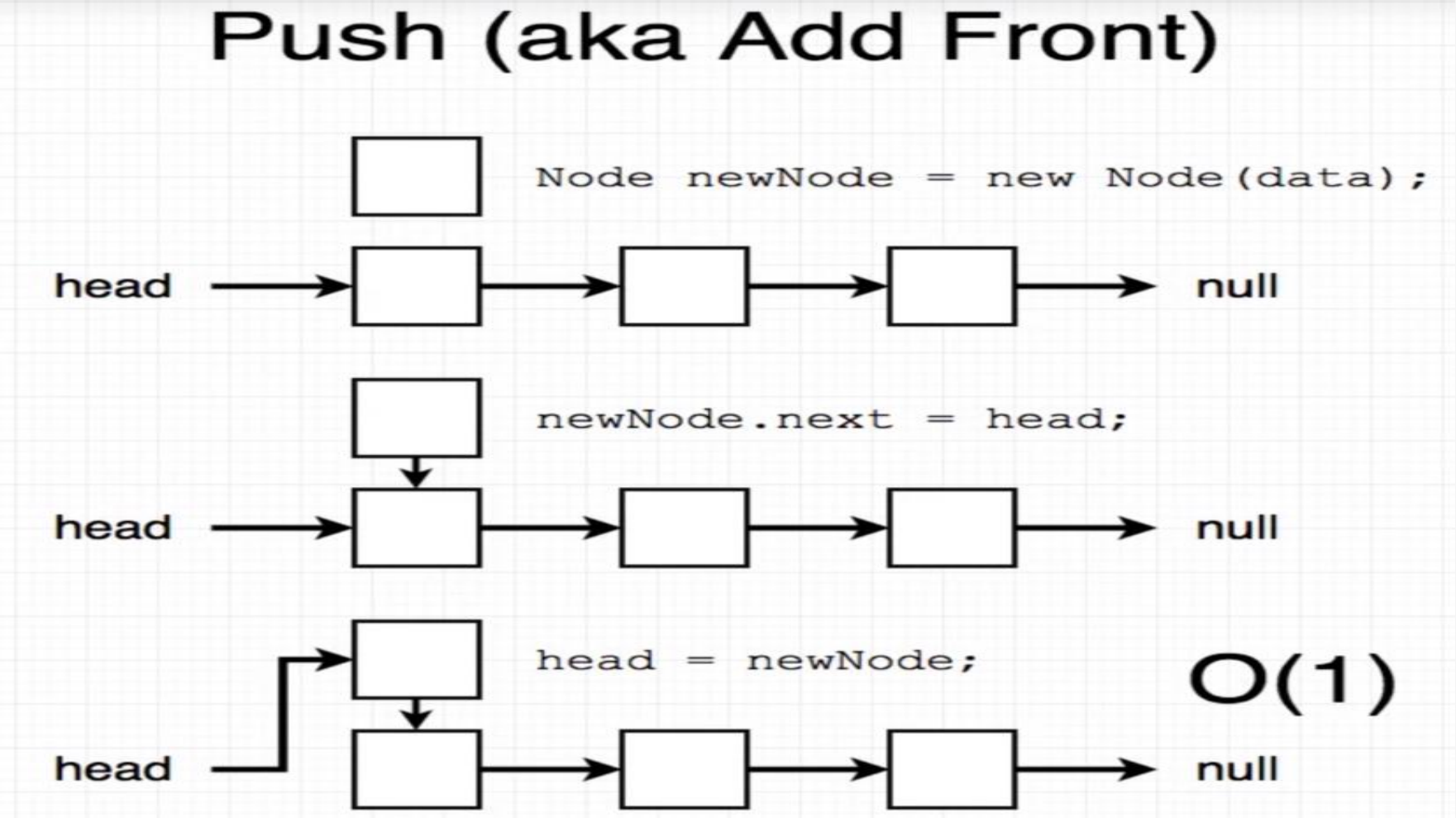
- void push(T data) -dodanie na stosie elementu.
- T pop() - pobranie elementu i usunięcie elementu ze stosu.
- T peek() - pobranie elementu , bez usunięcia

Stos – operacja push

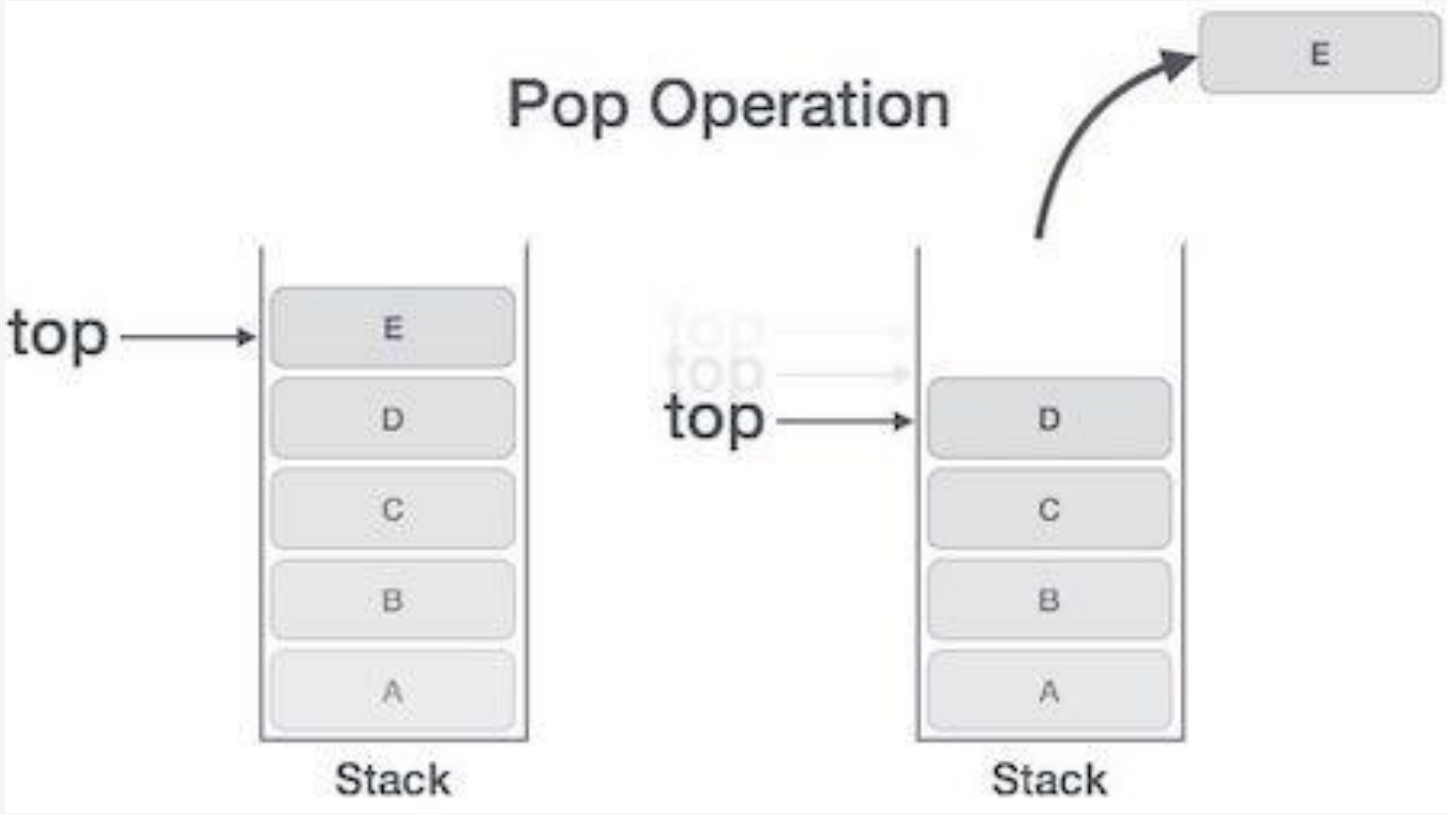


Zródło: https://www.tutorialspoint.com/data_structures_algorithms/images/stack_push_operation.jpg

Stos – operacja push

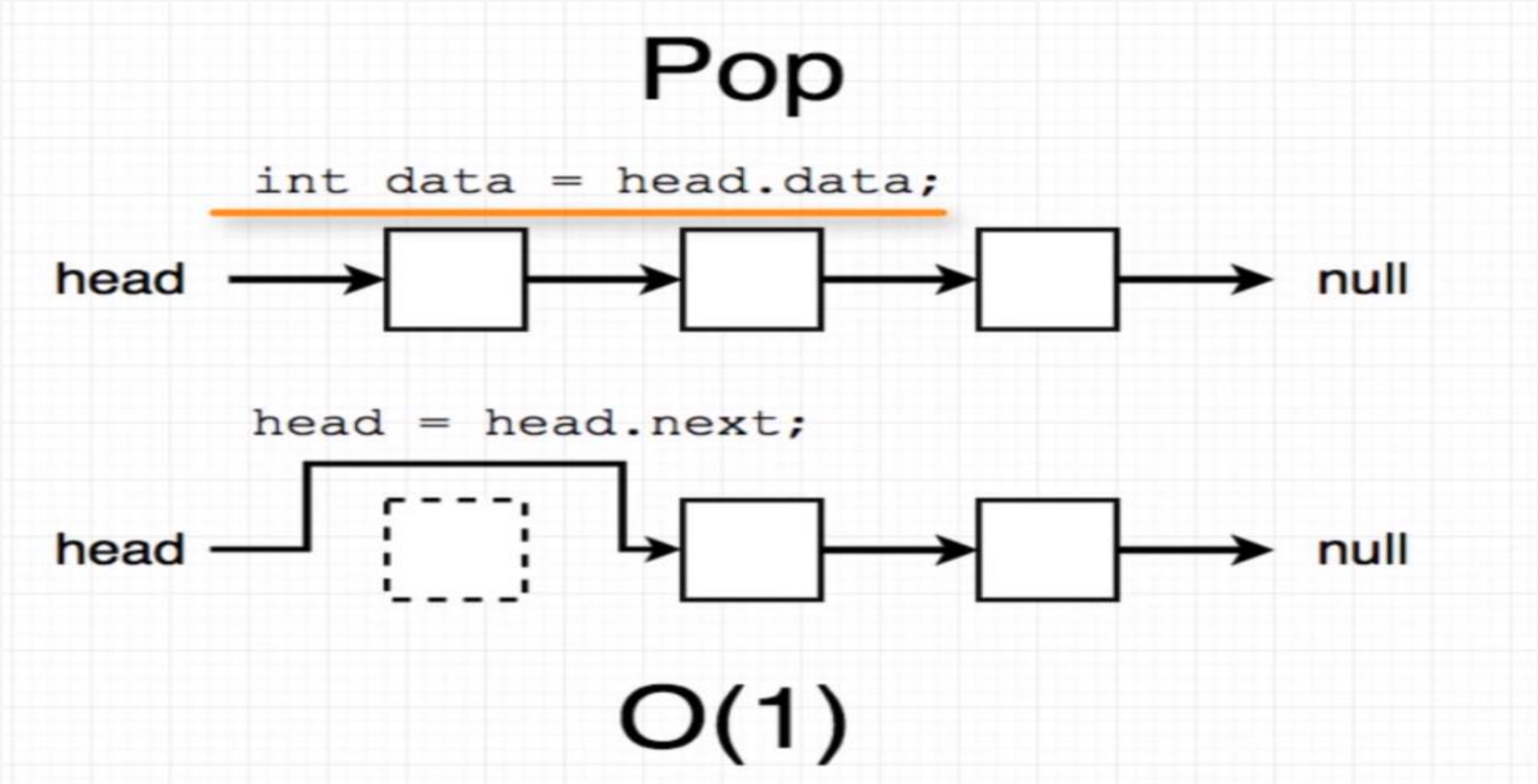


Stos – operacja pop



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

Stos – operacja pop



Stos – zastosowanie

- Edytor word/paint i operacja cofnij
- Pamięc Javy wykorzystuje stos do przechowywania ramek wywołan metod
- Operacja "cofnij" w przeglądarce

Stos – zadanie

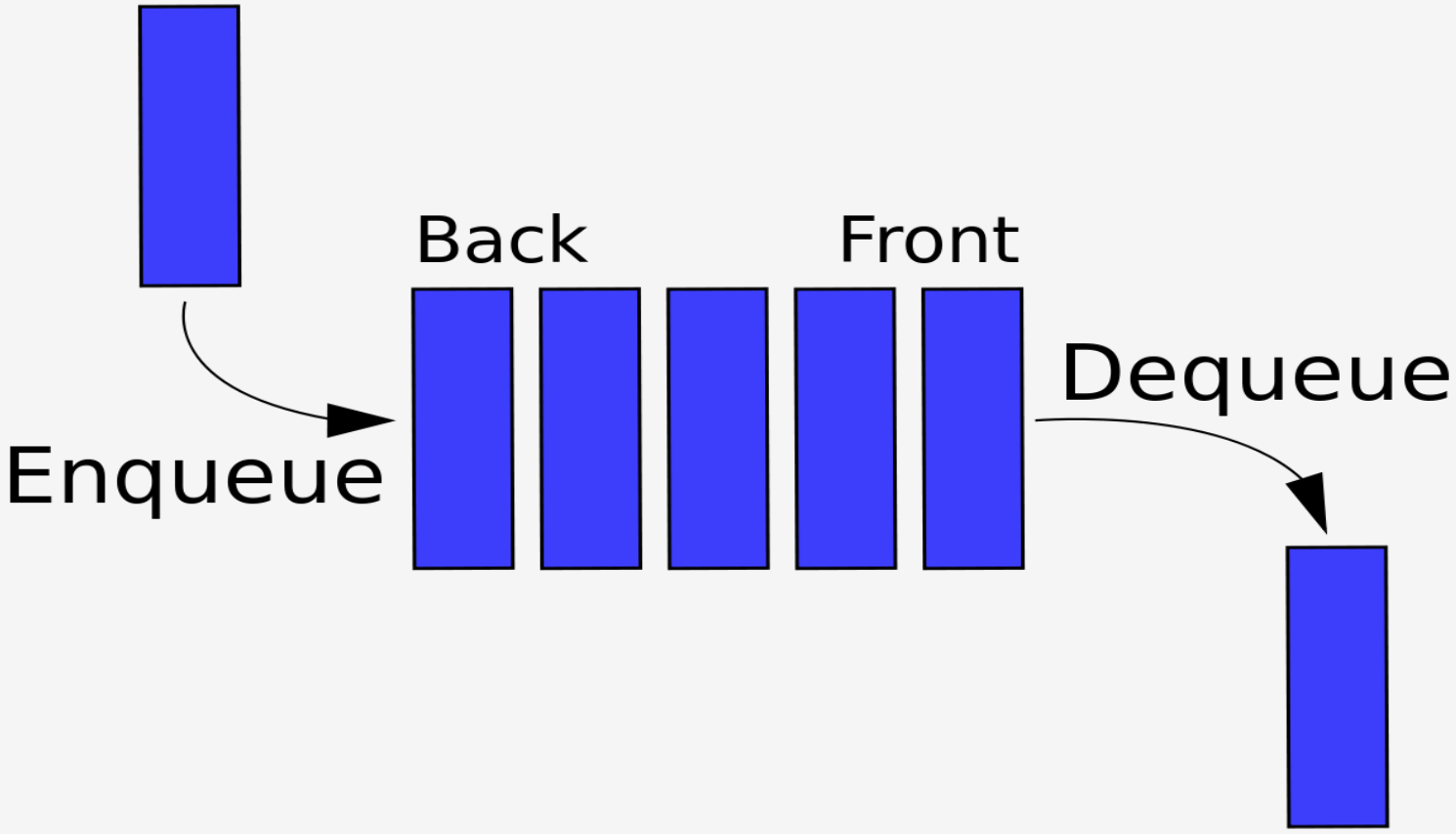
- Dokończ implementacje metody **`printStack()`** w klasie MyStack

- Napisz program, którego zadaniem będzie odwracanie słowa podanego przez użytkownika z użyciem klasy MyStack (Klasa **RevertWordWithStack**)

Kolejka

Kolejka(ang. Queue) - jest liniową listą, w której elementy są dodawane na jednym końcu, a usuwane na drugim. Przykładem kolejka w sklepie, gdzie dołączamy na końcu a wychodzimy z niej na początku.

Kolejka



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

Kolejka – zastosowanie w informatyce

Przykładowo kilka osób chce coś wydrukować na drukarce sieciowej. Ponieważ drukarka może wykonać tylko jedno zadanie w danej chwili, wszystkie pozostałe zostaną umieszczone w kolejce

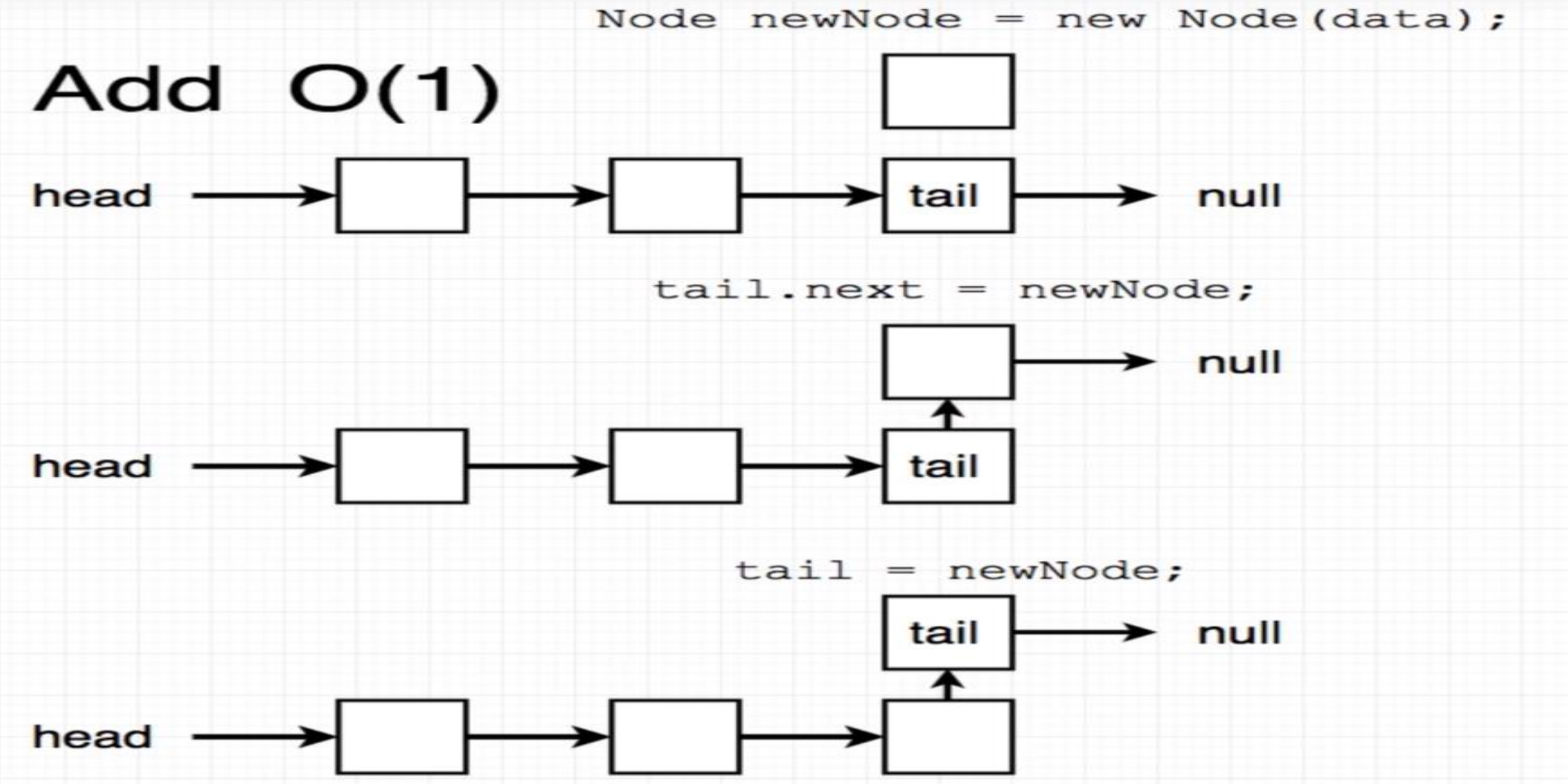
Kolejka – zastosowanie w informatyce

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

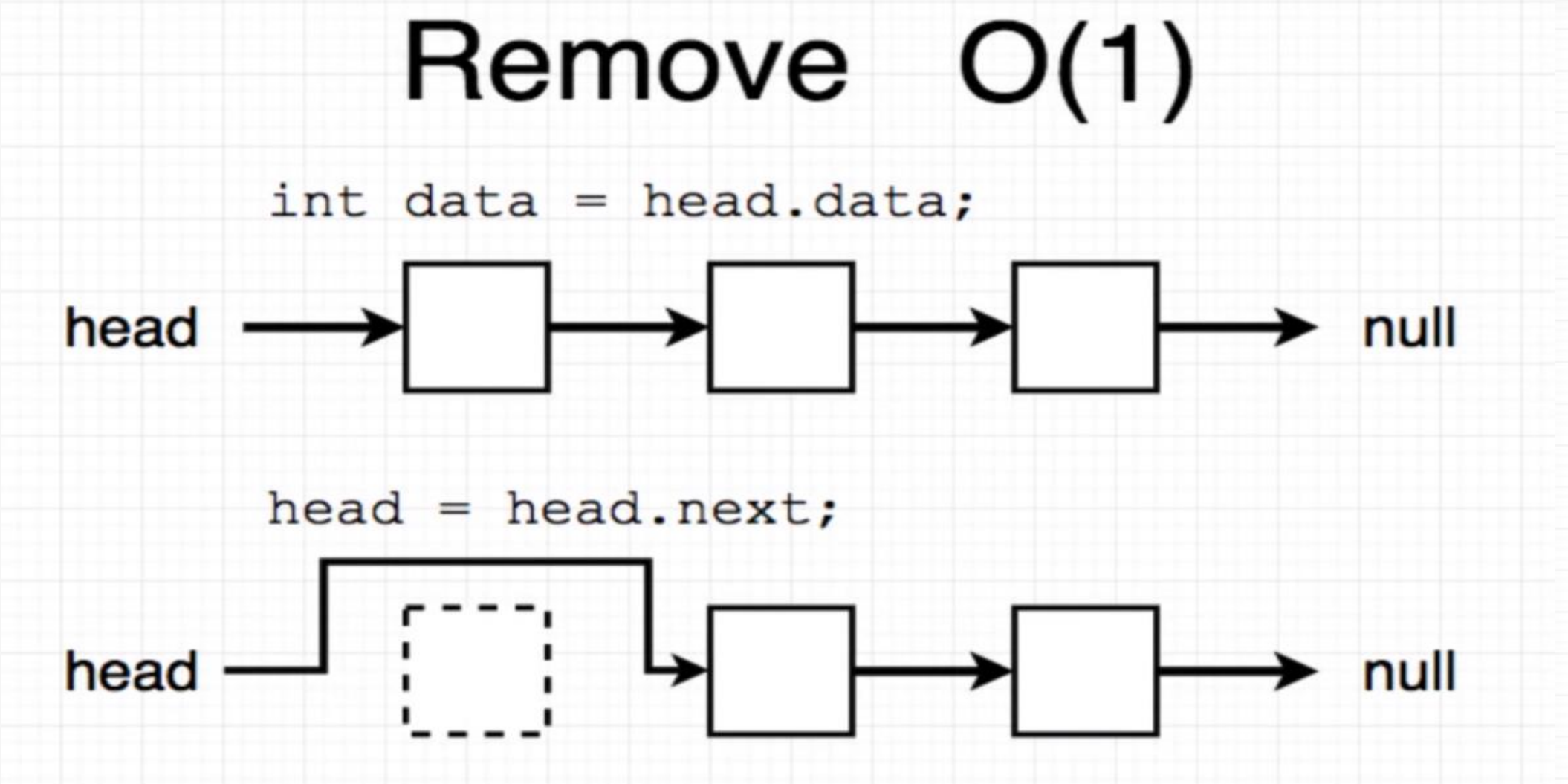
Lista operacja na kolejkach:

- Dodawanie elementu do kolejki(**`add`**),
- Pobranie i usunięcie elementu z kolejki(**`remove`**)

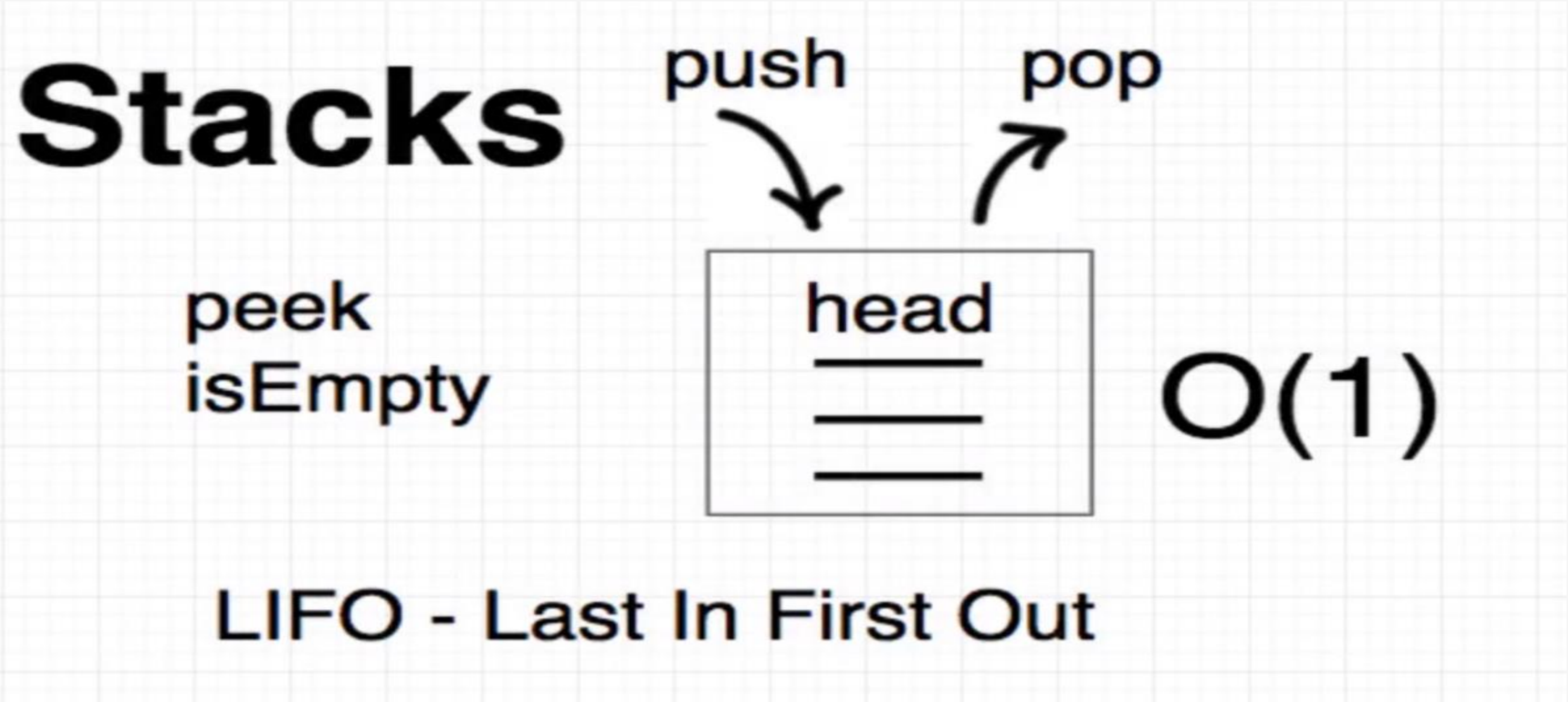
Kolejka – operacja add



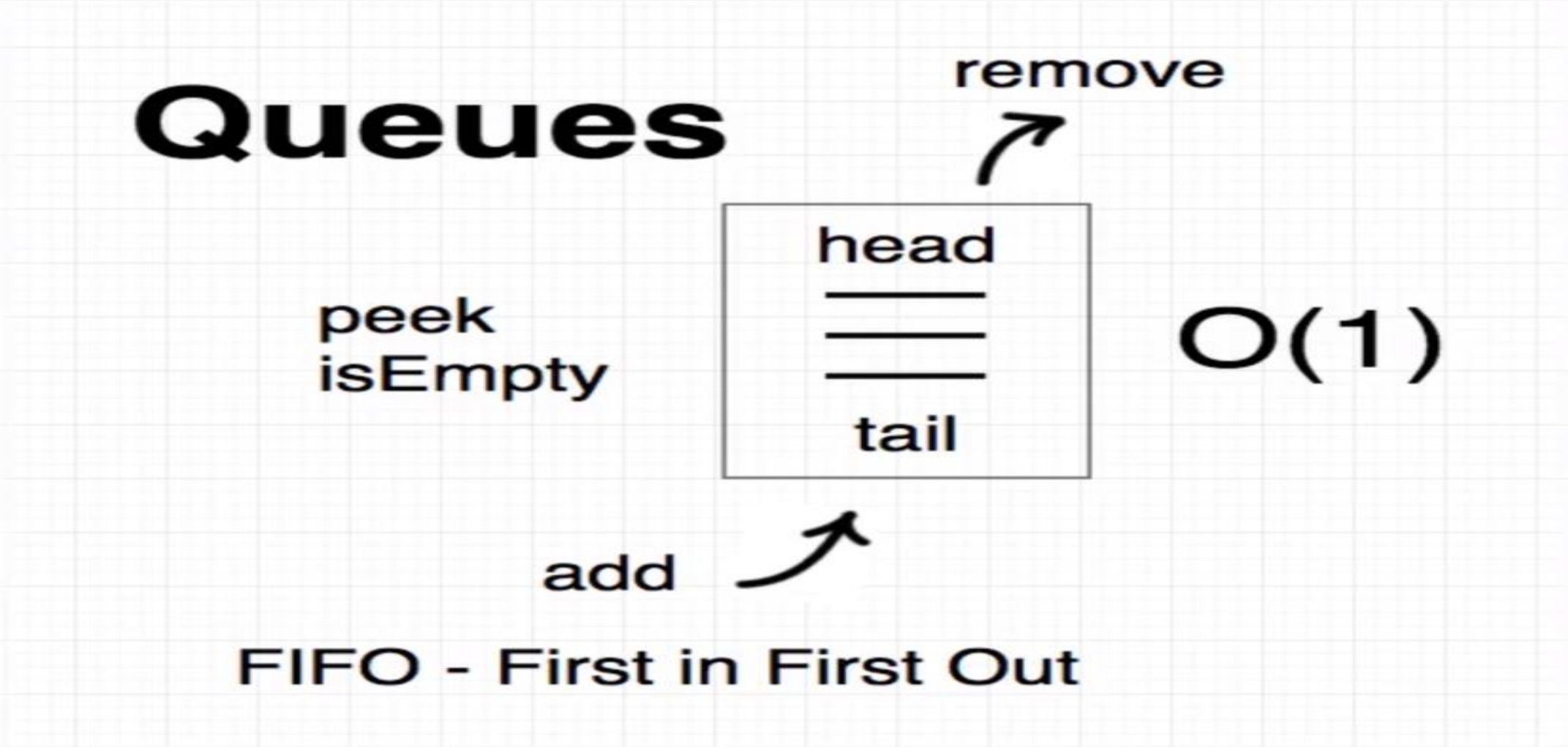
Kolejka – operacja remove



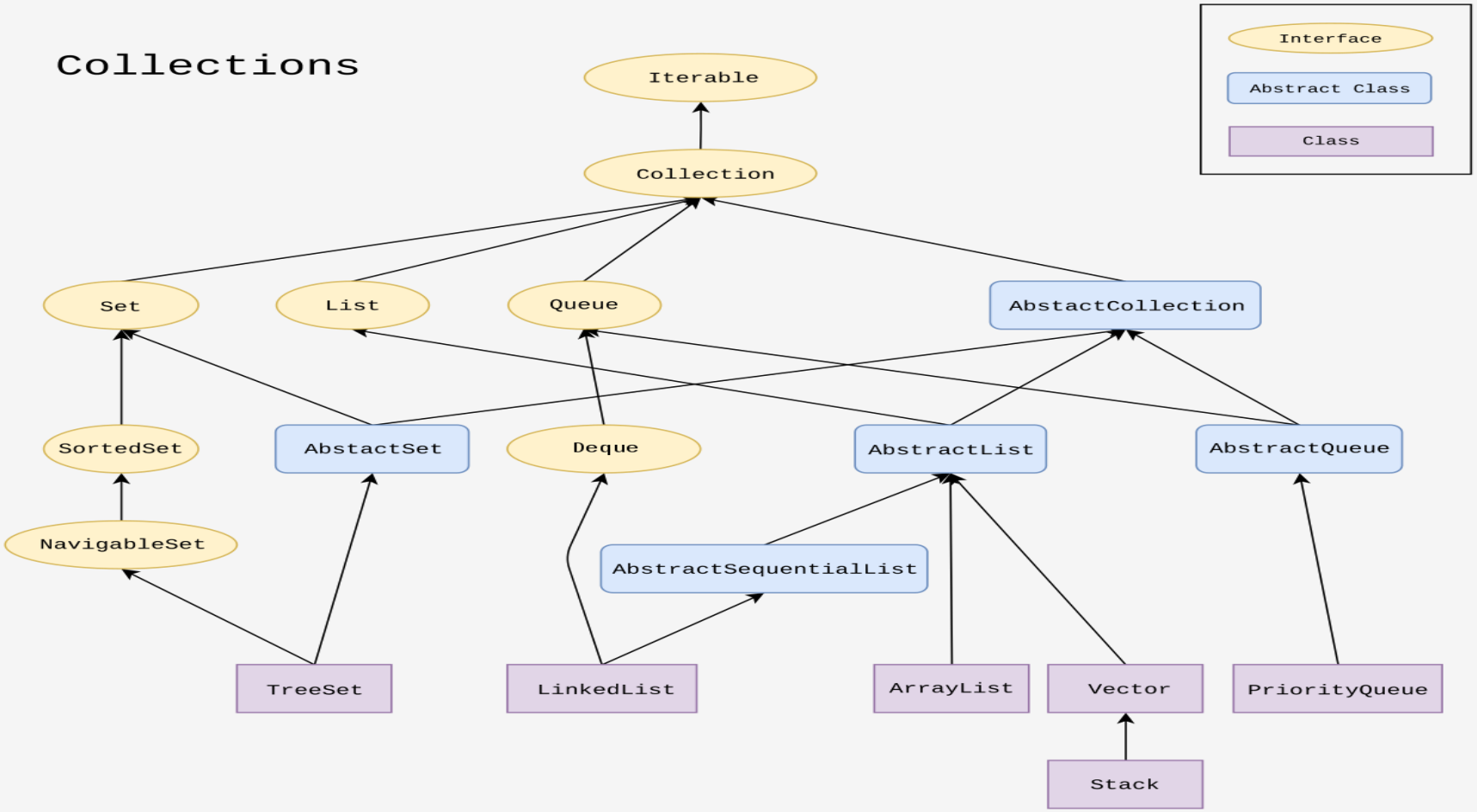
Stos co należy wiedzieć



Kolejka co należy wiedzieć



Java Collection – gotowe implementacje



Struktury danych w Javie: Interfejs Collection

Interfejs java.util.Collection to interfejs, na któym bazują kolekcje(oprócz Map). Definiuje on operacje, które mogą być wspierane przez wszystkie z nich, czyli:

- Dodanie element : **add,addAll**
- Usuwanie elementu: **remove,removeAll**
- Sprawdzenie, czy kolekcja zawiera określony element: **contains,containsAll**
- Zwrócenie rozmiaru(ilości elementó) kolekcji: **size**

Struktury danych : Typy generyczne(ang. Generic types)

- Typy generyczne to szablony
- Unikamy niepotrzebnego rzutowania
- Pozwalają na tworzyć bardziej elastyczne konstrukcje
- Java Collection Framework używa typów generycznych

Struktury danych w Javie: Interfejs java.util.List

Interfejs java.util.List zwiera wspólne funcjonaloności dla uporządkowanych kolekcji(ciągów). Kolekcje bazująca na tym interfejsie umożliwiają:

- Przecyzyjną kontrolę nad porządkiem elementów w kolekcji
- Dostęp do elementu na podstawie jego indeksu: **get,set**
- Wyszukanie elementu i zwrócenie jego indeksu: **indexOf**

Warto pamiętać:

- Listy, podobnie jak tablice indeksowane są od zera
- Typowe implementacje zezwalają na istnienie duplikatów elementów, ale z technicznego punktu widzenia nie jest to wymagana charakterystyka

Struktury danych w Javie: Interfejs java.util.Set

Interfejs **java.util.Set** zawiera wspólne funkcjonaloności dla nieuporządkowanych kolekcji(zbiorów). W przeciwieństwie to interfejsu **List**, **Set** posiada metody nakładające dodatkowe ogarniczenia:

- Zbiór nie może posiada dwóch jednakowych(w sensie metody equals) elementów.
- Może posiadać conajwyżej jeden element **null**

Struktury danych w Javie: Interfejs java.util.Set

Rozszerzenie interfejsu Set jest SortedSet(a jego z kolei NavigableSet). SortedSet przechowuję elementy w określonym porządku (naturalny lub zdefiniowany przez komparator) i definiuje następujące metody:

- **first()** - zwraca najmniejszy element w zbiorze
- **last()** - zwraca największy element w zbiorze

Struktury danych w Javie: Interfejs java.util.Set - Implementacje

Przykładowe implmentacje zbiorów:

- HashSet - zbiór oparty na tablicy haszującej**. Implementacja ta zezwala na umieszczenie **null** w zbiorze i nie potrządkuje elementów w żaden specyficzny sposób.
- TreeSet - zbiór operaty na drzewie czerwonoczarnym**. Elementy są przechowywane w naturalny porządku.Nie zezwalamy na umieszczenie **null(Co zaskakuje)**

Java Interfejs Comprator<T>

- Służy do porównywania obiektów w Javie - choćby tych zdefiniowanych przez Nas
- Klasa która implementuje interfejs ***Comparator<T>*** musi zaimplementować metodę ***int compare(T first, T second),*** która zwraca :
  - Liczbę >0 if first > second
  - Liczbę <0 if first < second
  - Liczbe ==0 if first == second

Kolejka Priorytetowa w Javie

- W Javie klasa **PriorytyQueue<T>** każdy element zanim zostanie dodany do kolejkcji, zostanie porównany za pomocą obiektu Compratator<T> przekazanego jako parametr do konstruktora np:

```
PriorityQueue<Bilet> pasazerowieSamolotu =
        new PriorityQueue<>(new PorownywarkaBiletow());
```

Kolekcja(Set/List/Queue/Stack) - dobierz kolekcję

- Ludzie stojący do lekarza
- Kolejność odkładania i mycia naczyń
- Konstrukcja i dekonstrukcja piramidy Cheopsa
- Slajdy w prezentacji
- Ubrania w szafie

Autor:

Zzłożność obliczeniowa na listach i zbiorach

Struktura	add	get	remove	contains	struktura danych
ArrayList	O(1) (amortyzowany)	O(1)	O(n)	O(n)	tablica
LinkedList	O(1)	O(n)	O(1)	O(n)	lista dowiązana
HashSet	O(1)	N/A	O(1)	O(1)	tablica haszująca
TreeSet	O(log n)	O(log n)	O(log n)	O(log n)	drzewo czerwono-czarne

Kolejka priorytetowa – zadanie(kolejka.pdf)

Twoim zadaniem jest napisanie aplikacji służącej do decydowania o kolejności przyjęć w szpitalu.

Aplikacja powinna pozwalać na:

- Rejestrację nowego pacjenta
- Pobieranie następnej osoby z kolejki(która wejdzie do lekarza)
- Podglądanie kto jest następny w kolejce

Wykonaj następujące czynność:

1. W klasie *Patient* dodaj pola
   - name(*imię*), surname(*nazwisko*), howAngry(*jak bardzo zły*),diagnozed(enum z polem zarazliwość typu Disease - już istnieje). Pamiętaj o getterach i setterach(chyba że tworzysz jako immutable)
2. W klasie *HospitalQueueService* zaimplementuj metody:
   - add() – dodaj pacjenta do kolejki
   - next() - pobierająca pacjenta z kolejki(usuwa)
   - howIsNext() - Zwraca pacjenta następnego(nie usuwa)
3. Uzupełnij implementacje metody *PatientComparator.java* według algorytmu:
   - W pierwszej kolejność powinny być obsługiwane osoby z czymś poważnym(Diease.SOMETHIN_SEROIUS)
   - Dalej osoby których iloczyn howAngry i zarazlwość będzie wyższy
4. W klasie *PatientGenerator* ustaw pola na obiekcie Patient

Autor:

Odrzwarzacz muzyczny – zadanie(player.pdf)

Zapoznaj się z plikiem player.pdf, zawierającym pełną treść zadania

Autor:

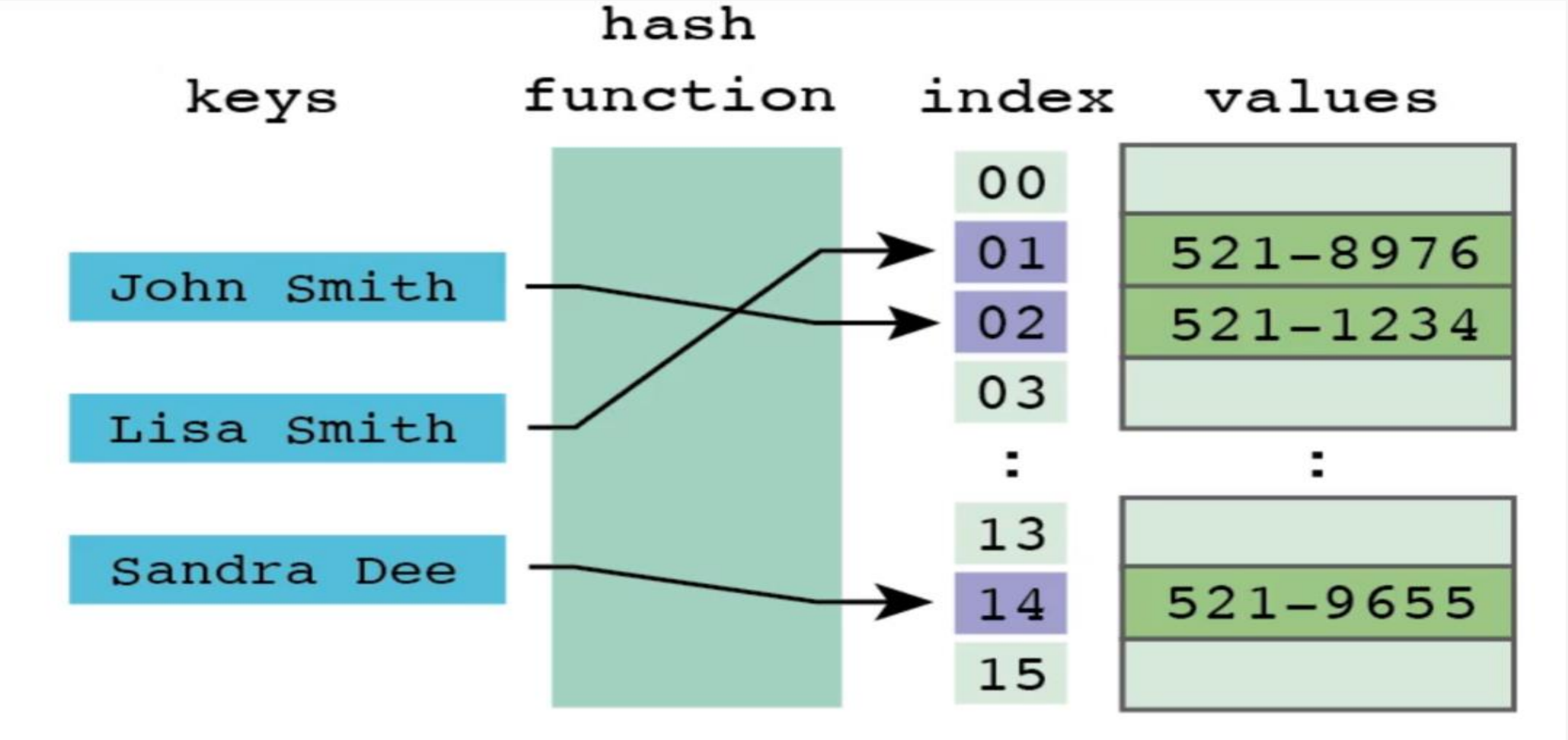
Haszowanie(ang. Haszing)

Jest metodą super szybkiego wyszukiwania danych w tablicach.

Polega na tym że mamy tzw. **Funkcję haszującą** (ang. Hash function), która dla danego zestawu danych tworzy liczbę zwaną haszem (ang. Hash).

Liczbę tą(hash) używamy jako indeks w **tablicy haszowanej (ang. Hash table)** do dostępu do danych

Haszowanie - wizualizacja



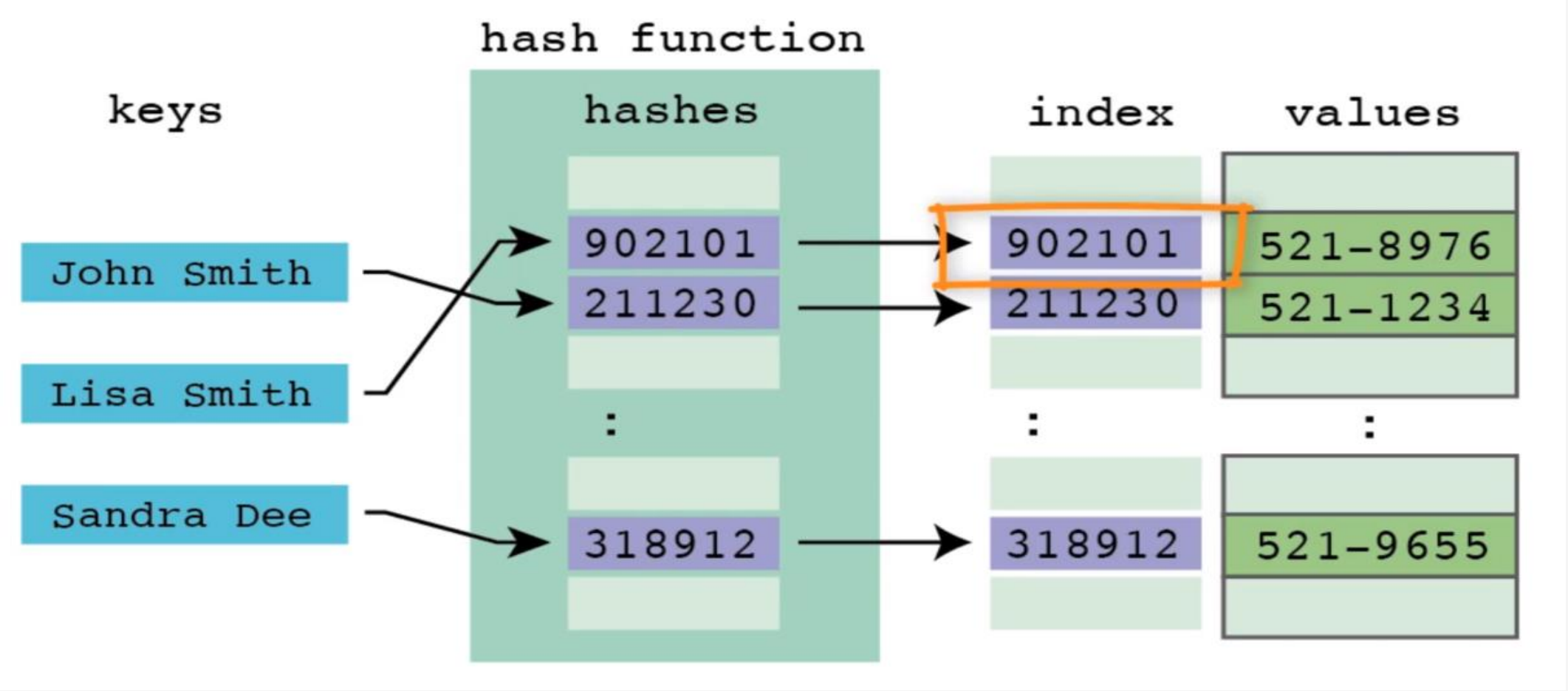
Autor:

Haszowanie - w ujęciu Javy

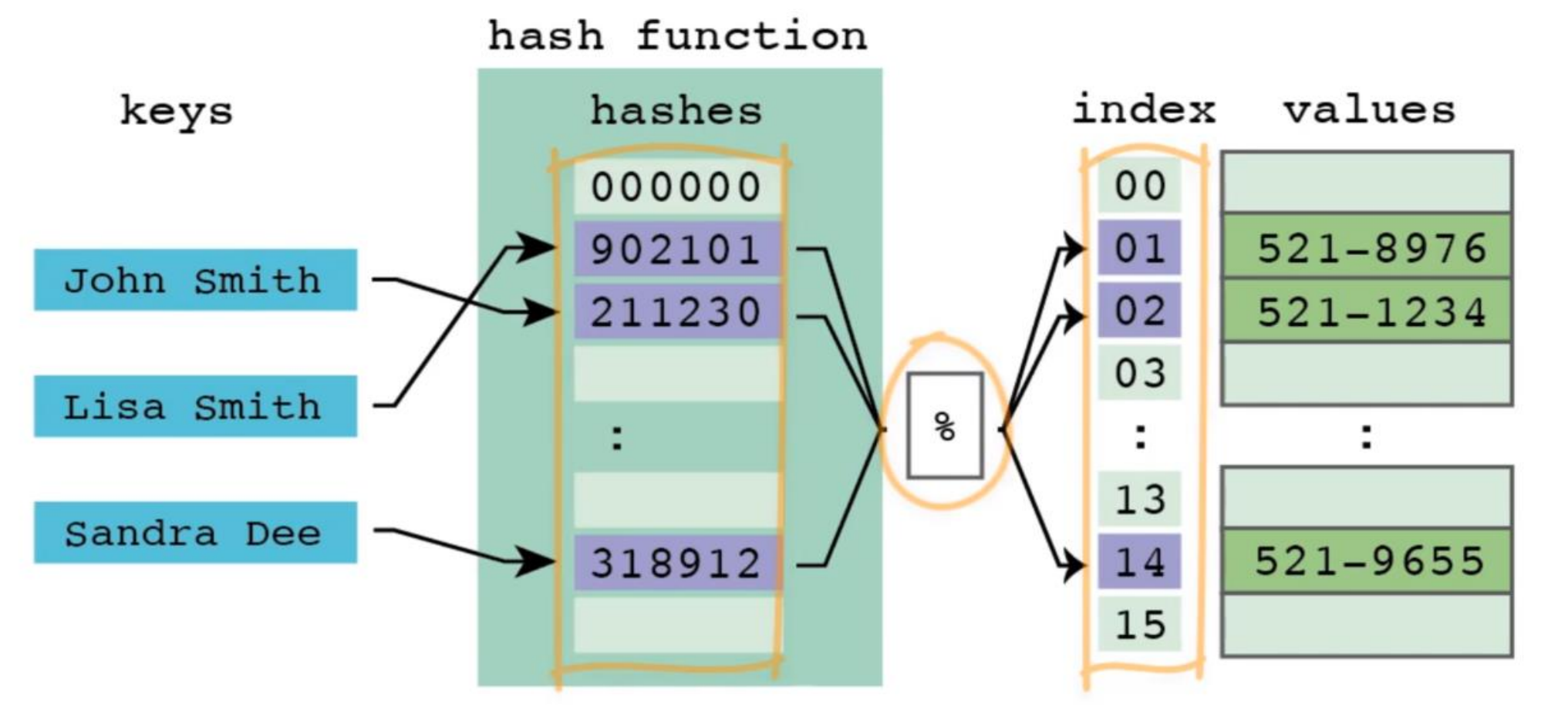
Hashcode – to numeryczna wartość reprezentująca obiekt

```
public class Person {
    private int id;
    private String address;

    public Person(int id, String address) {
        this.id = id;
        this.address = address;
    }

    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (address != null ? address.hashCode() : 0);
        return result;
    }
}
```

Haszowanie – konwersja indeksów



Haszowanie – konwersja indeksów



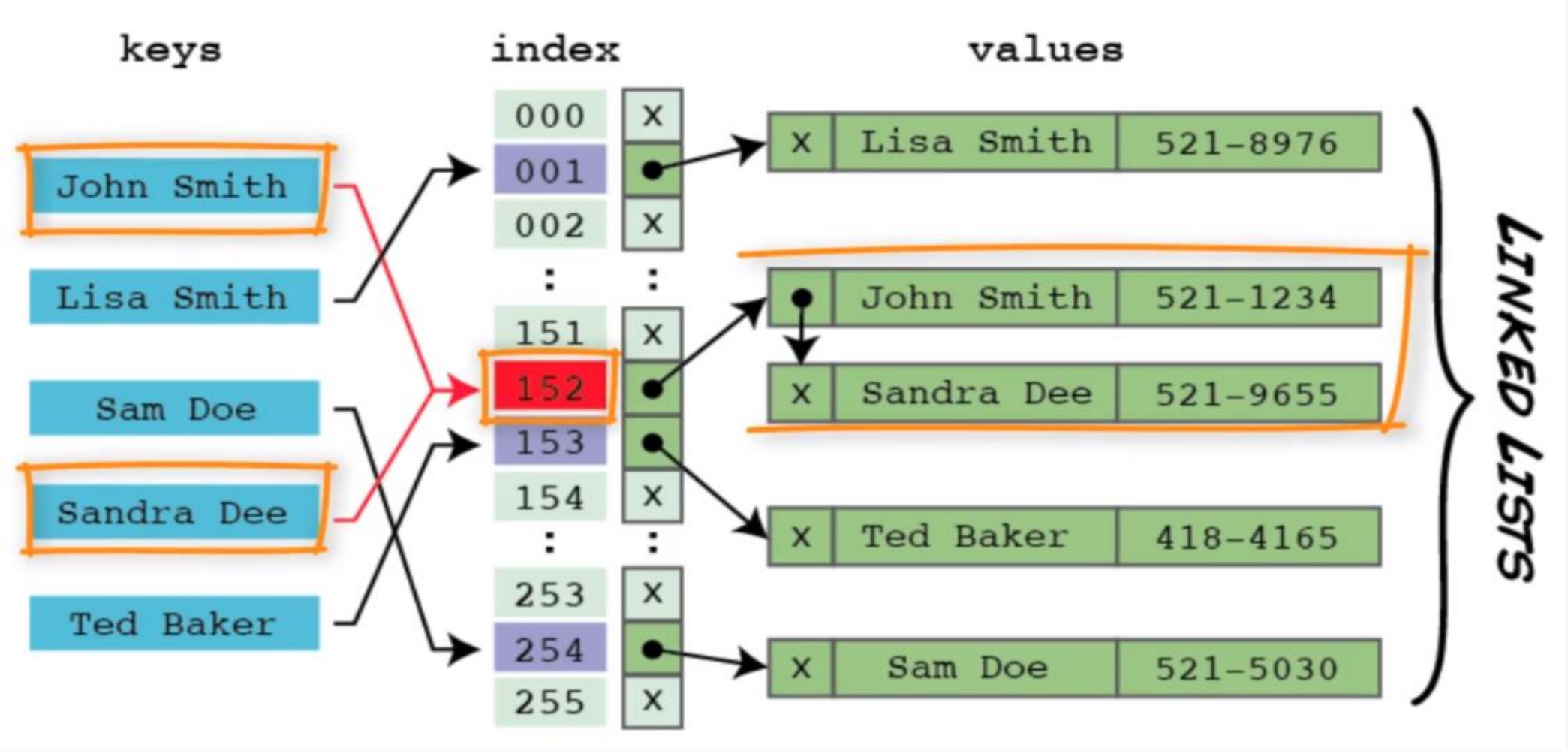
Haszowanie – konwersja indeksów operacja modulo

Haszowanie - problemy

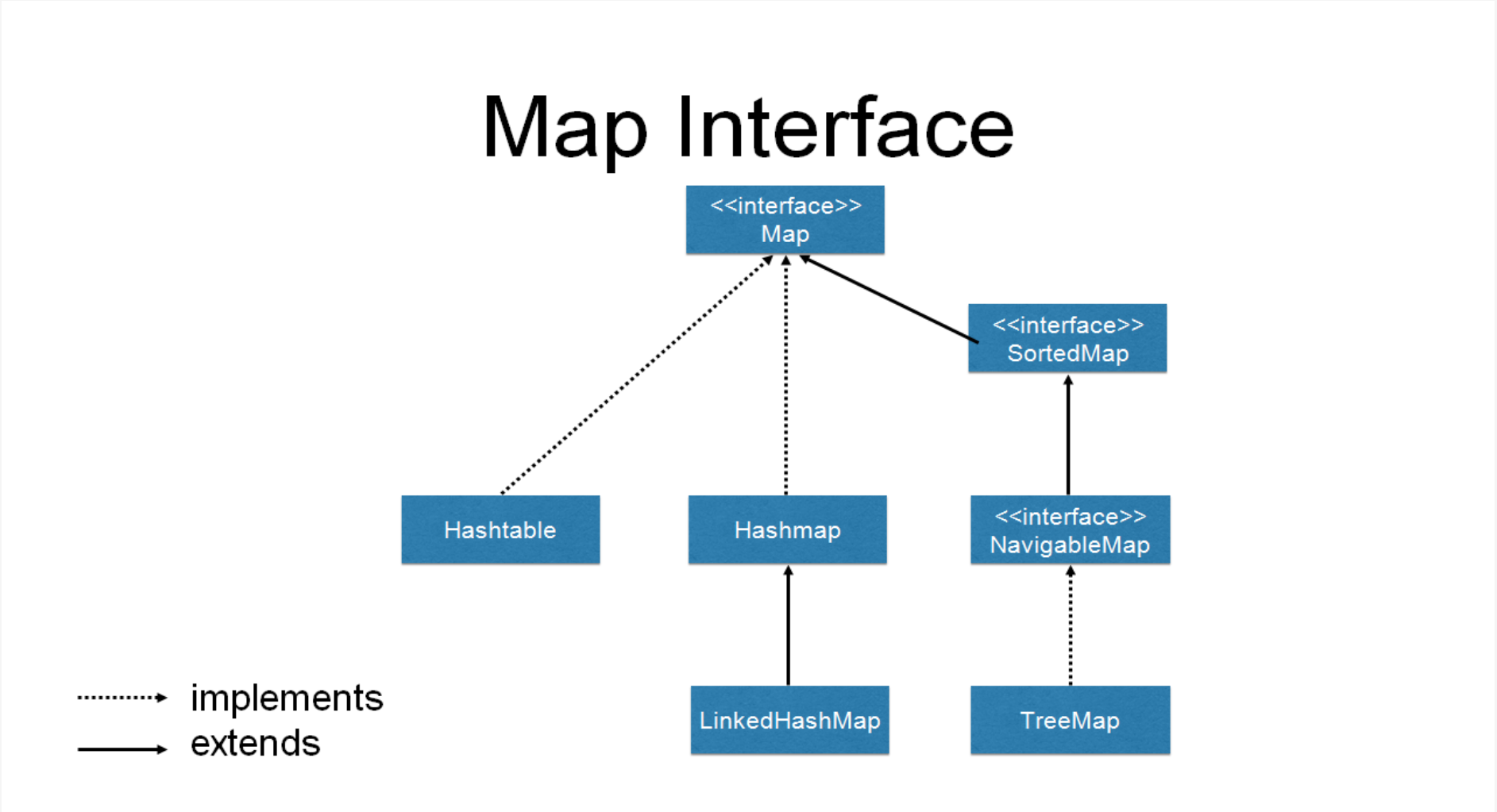
Podstawowym problem haszowania to **kolizyjność**.

Polega one na tym, iż funkcja haszująca tworzy te same wartości hashy(indeksów w tablicy haszującej) dla wielu róznych danych.

Haszowanie - problemy



Haszowanie – Interfejsy java.utl.Map i java.utl.SortedMap



Haszowanie – Interfejsy java.utl.Map i java.utl.SortedMap

Interfejs java.utl.Map zawiera wspólne funkcjonalności dla map łączących klucze z odpowiadającymi im wartościami.

Klucze przechowywane w mapie muszą być unikatowe. Dla map zdefniowane są następujące operacje:

- dodawanie/ustawianie elementów: ***put, putAll***
- odczytywanie elementów: ***get***
- sprawdzenie, czy dany klucz lub wartość znajduje się w mapie: ***containsKey***, ***containsValue***
- usunięcie danego klucza: ***remove***
- pobranie ilości elementów znajdujących się w mapie: ***size***
- utworzenie kolekcji z kluczami, wartościami i ich parami: ***keySet, values, entrySet***
- tworzone w ten sposób kolekcje są tak naprawdę widokami, korzystają z danych przechowywanych w mapie, a nie kopiują ich.

Haszowanie – Interfejsy java.utl.Map i java.utl.SortedMap

Interfejs Map jest rozszerzany przez interfejsy SortedMap i NavigableMap, będące dla mapy tym czym SortedSet i NavigableSet dla zbioru.

Przykładowe implementacje:

- **HashMap**: nieuporządkowana mapa
- **TreeMap**: NavigableMap, bazująca na drzewie czerwono-czarnym
- **LinkedHashMap**: jak HashMap, ale przechowuje podwójnie dowiązaną listę kluczy – umożliwia to odtworzenie porządku w jakim był wstawiane podczas iterowania.

Złozoność obliczeniowa na mapach

Struktura	get/put	containsKey	struktura danych
HashMap	O(1)	O(1)	tablica haszująca
TreeMap	O(log n)	O(log n)	drzewo czarne
LinkedHashMap	O(1)	O(1)	tablica haszująca + lista dowiązana

Haszowanie – co należy pamiętać

- Znać wydajność operacji (put/get)
- Obiekty Immutable jako klucz
- Pojęcie haszowania w ujęciu HashMap<K,V> w javie
- Kolizje i co się dzieje w przypadku wystąpienia
- Opisać konwersję indeksu, czyli że hashCode za pomocą funkcji haszującej jest konwertowany do indeksu
- Znać strukture mapy czyli obiekt HashEntry
  - Widzieć że przechowuje i klucz i wartość

Mapy zadanie (bank_account.pdf)

- Znać wydajność operacji (put/get)
- Obiekty Immutable jako klucz
- Pojęcie haszowania w ujęciu HashMap<K,V> w javie
- Kolizje i co się dzieje w przypadku wystąpienia
- Opisać konwersję indeksu, czyli że hashCode za pomocą funkcji haszującej jest konwertowany do indeksu
- Znać strukture mapy czyli obiekt HashEntry
  - Widzieć że przechowuje i klucz i wartość