Programowanie I

Jakub Płonka

Wprowadzenie

1. Przedstaw się
2. Dlaczego wybrałaś/eś ten kurs?
3. Jak wyobrażasz sobie pracę programisty?
4. Czego oczekujesz od tego bloku?
5. Jaki sposób pracy Ci odpowiada?

Autor: Jakub Płonka

Czym się zajmiemy

1. Powtórka poznanych już elementów Java
2. Oswojenie się z GITem
3. Schemat blokowy
4. Złożoność obliczeniowa
5. Algorytmy:
    1. Struktury danych - single linked list, double linked list, array list, queue, stack, prioritized queue
    2. Wyszukiwanie - linear sort, binary sort
    3. Sortowanie - bubble sort
6. Z czym z tego bloku i nie tylko możecie spotkać się na rekrutacjach - wplecione pomiędzy zagadnienia

Autor: Jakub Płonka

Rozgrzewka

Zadania

1. Użytkownik podaje wysokość i szerokość, narysuj szachownicę o zadanych wymiarach
2. Użytkownik podaje bok kwadratu, który wypełniony jest w następujący sposób:

```
oooxxx
xxxooo
```

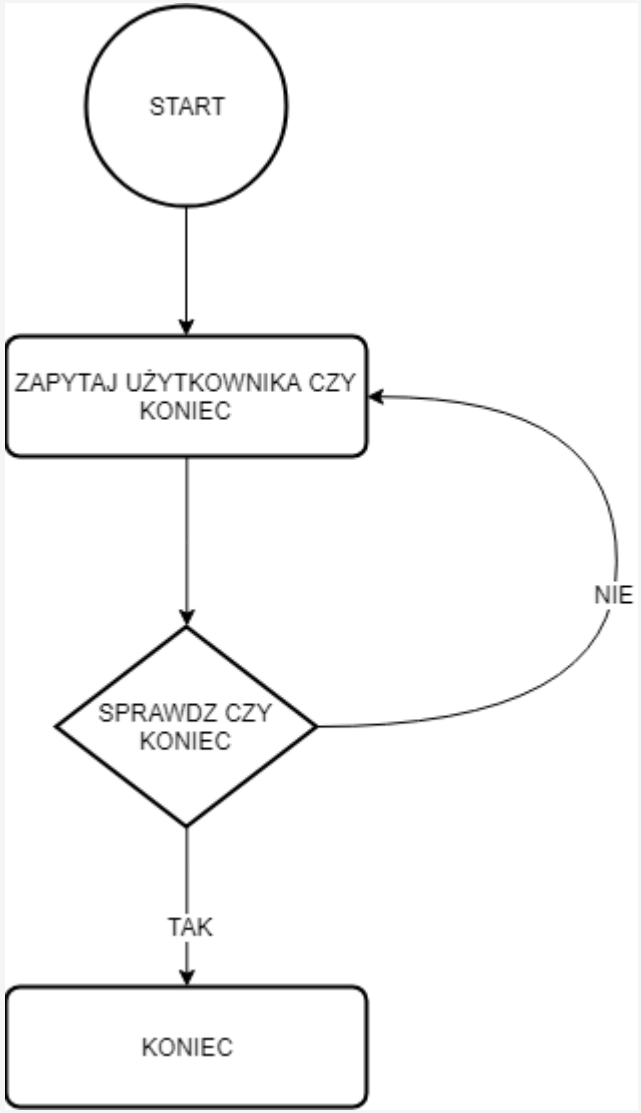
- *Dodaj obramowanie do kwadratu powyżej:

```
........
.oooxxx.
.xxxooo.
........
```

3. Stwórz klasę człowiek mającą pola: imię, wiek, płeć (pamiętaj o nadaniu odpowiednich typów, może będzie trzeba stworzyć jakiegoś enuma?):
   - Stwórz listę, która będzie zawierała następujące osoby:
     - Adam, wiek 15
     - Basia, wiek 30
     - Paweł, wiek 20
     - Kasia, wiek 25
     - Marcin, wiek 101
   - Wypisz:
     - osobno mężczyzn i osobno kobiety
     - osoby pełnoletnie
     - wszystkie osoby, których wiek jest wielokrotnością 10
     - zsumuj wiek wszystkich osób
     - wszystkie osoby mające “si” w imieniu

Autor: Jakub Płonka

Schemat blokowy



Autor: Jakub Płonka

Zadanie

Narysujmy wspólnie schematy blokowe, które następnie zaimplementujecie:

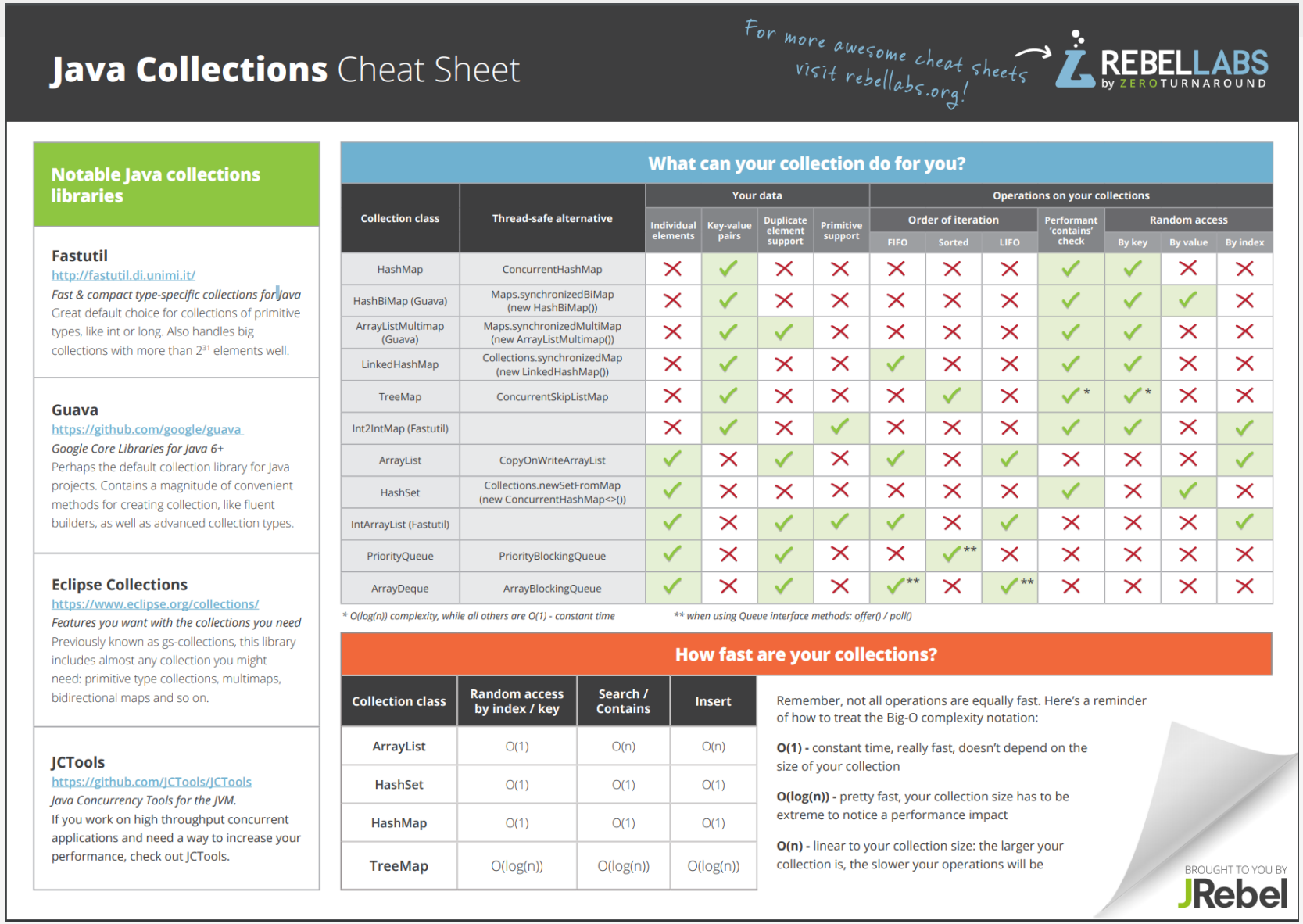
1. Palindrom - *Palindrom (gr. palindromeo – biec z powrotem) – wyrażenie brzmiące tak samo czytane od lewej do prawej i od prawej do lewej. Przykładem palindromu jest: Kobyła ma mały bok - https://pl.wikipedia.org/wiki/Palindrom, dostęp 11.11.2018,* inne przykłady: Ala, Aga, Bob, kajak, potop, zaraz
2. Anagram - *nazwa wywodząca się od słów greckich: ana- (nad) oraz grámma (litera), oznacza wyraz, wyrażenie lub całe zdanie powstałe przez przestawienie liter bądź sylab innego wyrazu lub zdania, wykorzystujące wszystkie litery (głoski bądź sylaby) materiału wyjściowego. - https://pl.wikipedia.org/wiki/Anagram, dostęp 11.11.2018,* przykład: markotny - romantyk
3. Wyznacz największą liczbę z podanej przez użytkownika listy

Autor: Jakub Płonka

Kolekcje w Java - co już znamy?

Autor: Jakub Płonka

Kolekcje w Java



Java Collections Cheat Sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Notable Java collections libraries

Fastutil
http://fastutil.di.unimi.it/
Fast & compact type-specific collections for Java
Great default choice for collections of primitive types, like int or long. Also handles big collections with more than 2^{31} elements well.

Guava
https://github.com/google/guava
Google Core Libraries for Java 6+
Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

Eclipse Collections
https://www.eclipse.org/collections/
Features you want with the collections you need
Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimaps, bidirectional maps and so on.

JCTools
https://github.com/JCTools/JCTools
Java Concurrency Tools for the JVM.
If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

What can your collection do for you?

Collection class	Thread-safe alternative	Your data				Operations on your collections						
						Order of iteration				Random access		
		Individual elements	Key-value pairs	Duplicate element support	Primitive support	FIFO	Sorted	LIFO	Performant 'contains' check	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	✓	✗	✓	✗	✓**	✗	✓**	✗	✗	✗	✗

** O(log(n)) complexity, while all others are O(1) - constant time*

*** when using Queue interface methods: offer() / poll()*

How fast are your collections?

Collection class	Random access by index / key	Search / Contains	Insert
ArrayList	O(1)	O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
HashMap	O(1)	O(1)	O(1)
TreeMap	O(log(n))	O(log(n))	O(log(n))

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

O(1) - constant time, really fast, doesn't depend on the size of your collection

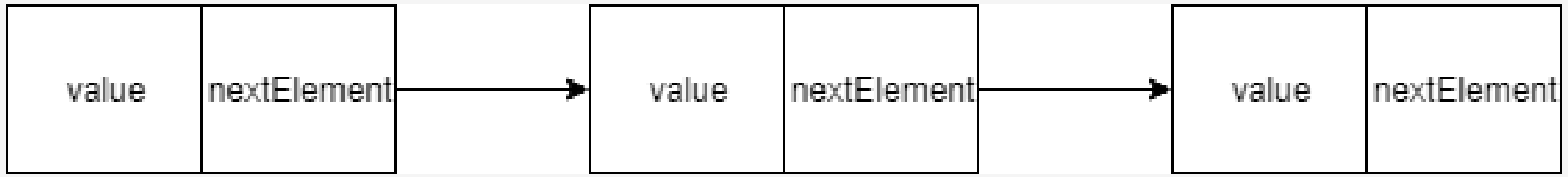
O(log(n)) - pretty fast, your collection size has to be extreme to notice a performance impact

O(n) - linear to your collection size: the larger your collection is, the slower your operations will be

BROUGHT TO YOU BY JRebel

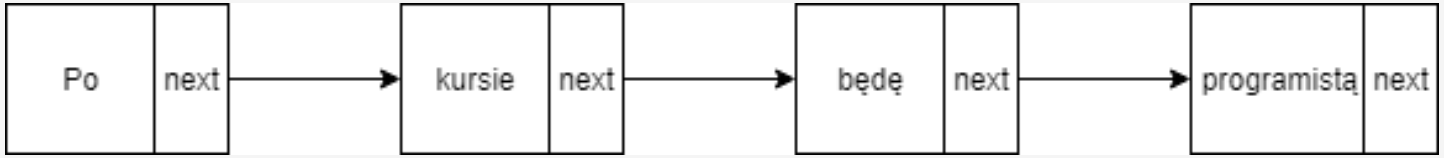
Autor: Jakub Płonka

Lista oparta o referencje - single linked list



Autor: Jakub Płonka

Pytania kontrolne



- Jak przejść z 1 elementu do 3?
- Co należy zrobić, żeby dodać element pomiędzy 2, a 3 elementem listy?
- Jak zmierzyć długość listy?
- Na co wskazuje ostatni element listy?
- **Jak znaleźć środek listy w jednym przebiegu?**
- Co znaczy, że taka struktura może się zapętlać?

Autor: Jakub Płonka

Zadania - uwaga

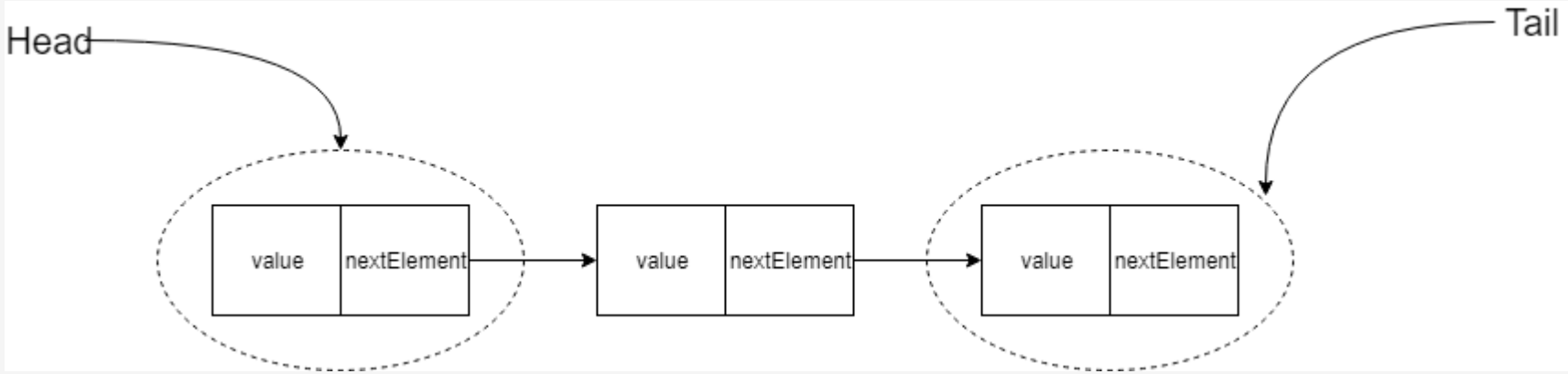
Od tej chwili korzystamy aktywnie z gita, załóżcie repozytorium w swoim ulubionym serwisie i commitujcie - nawet po jednym podpunkcie z zadania. W razie problemów chętnie pomogę ☺

Autor: Jakub Płonka

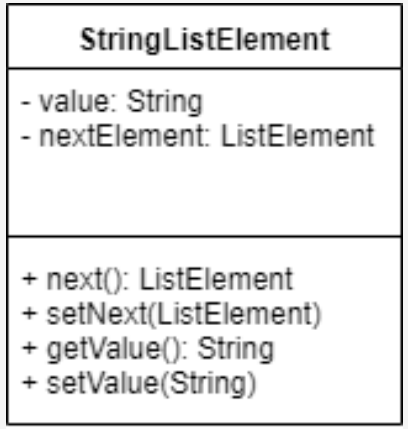
Co znaczy, że klasa jest Immutable?

Autor: Jakub Płonka

Zadanie - lista oparta o referencje - single linked list

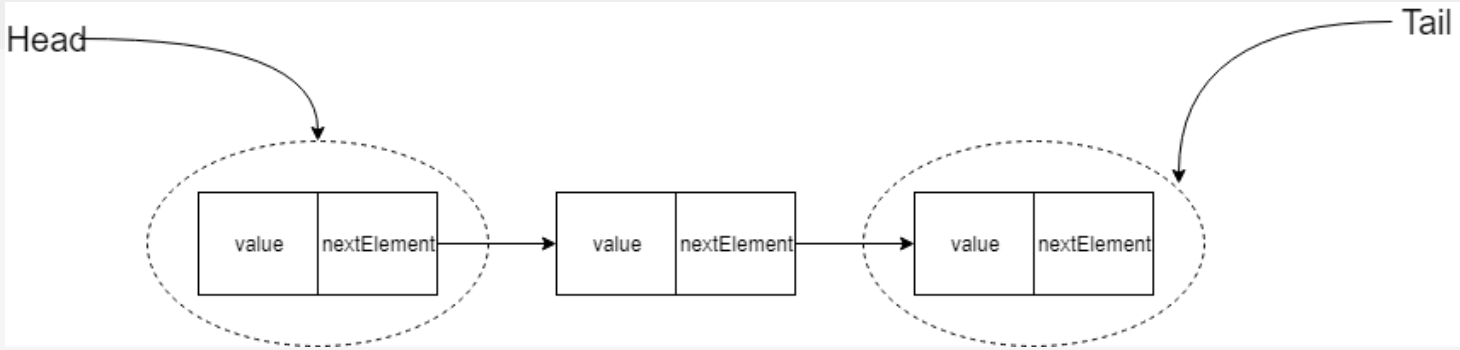


1. Zaimplementuj strukturę `StringListElement`
2. Dodaj 3 elementy: “Ala”, “ma”, “kota”
3. Wypisz wszystkie elementy zgodnie z kolejnością wstawienia
4. Wypisz wszystkie elementy od najdłuższego do najkrótszego (technika dowolna)
5. *Zmień nazwę klasy na `ListElement` i spraw, aby mogła przyjmować jako wartość obiekt dowolnego typu (wskazówka - generyki), następnie stwórz ponownie klasę o nazwie `StringListElement`, która będzie rozszerzała `ListElement`
6. *Zaimplementuj `ImmutableListElement` - zasada działania ta sama, jedynie odpadnie kilka metod, a dojdzie konstruktor ☺
7. *Odwróć kolejność elementów w liście



Autor: Jakub Płonka

Zadanie - lista oparta o referencje - single linked list



1. Zaimplementuj strukturę `StringList`
2. Dodaj 3 elementy korzystając jedynie ze `StringList`: “Java”, “jest”, “fajna”
3. Wypisz elementy zgodnie z kolejnością wstawienia
4. Wypisz elementy od najdłuższego do najkrótszego (technika dowolna)
5. Spraw, aby klasa `StringList` implementowała `Iterable`, a `StringListElement` - `Iterator`
6. Odfiltruj wszystkie element z listy z punktu 2 zaczynające się od j/J, pomocne może okazać się `StreamSupport.stream(SplitIterator)`
7. Wypisz elementy w odwrotnej kolejności
8. Co jeśli w trakcie iterowania jeden z elementów zostanie usunięty
9. *dodaj do StringList funkcję addAt(String, int), która przyjmuje również miejsce w które należy wstawić nowy element
10. *zmień nazwę klasy na `SingleLinkedList` i spraw, aby mogła przyjmować jako wartość obiekt dowolnego typu (wskazówka - generyki), następnie stwórz klasę `StringList` rozszerzającą `SingleLinkedList<String>`

StringListElement
- value: String - nextElement: ListElement
+ next(): ListElement + setNext(ListElement) + getValue(): String + setValue(String)

StringList
- first: ListElement - last: ListElement - ???
+ head(): ListElement + tail(): ListElement + add(String) + size(): long + remove(String) + get(long): String

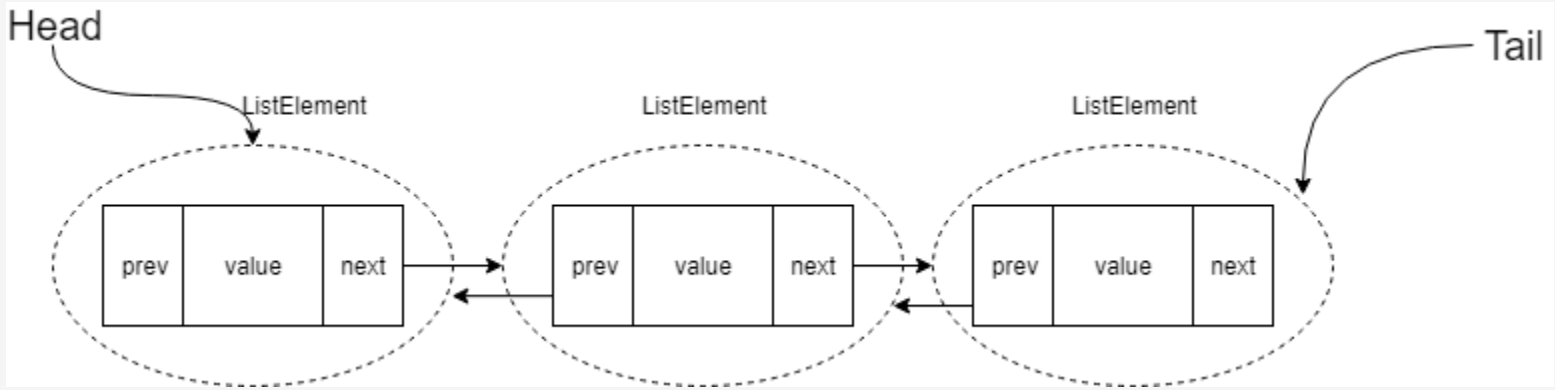
Autor: Jakub Płonka

Przerywnik - pytania na rozmowę rekrutacyjną

1. **Co wprowadzono w Javie 8? Co wprowadzono w Javie 9/10?**
2. Czym jest HashMap dla Map?
3. Co jest unikalne w commicie?
4. **Jak posortować listę?**
5. ***Co znaczy, że obiekt jest immutable?**
6. Czego lepiej użyć LocalDate czy Date?
7. Jaka jest kolejność elementów w HashSet?
8. Jak posortować Set
9. **Jakie są modyfikatory dostępu (4)**
10. **Co to jest klasa anonimowa?**
11. **Który blok w try catch finally na pewno zawsze się wykona**
12. **Jak sprawić, żeby się nie wykonał**
13. **Jakie są 2 podstawowe typy wyjątków**
14. Co robi słowko kluczowe this
15. **Co robi słowo kluczowe super**
16. Co się stanie kiedy na Stringu = null wywołamy .isEmpty()
17. **Czy Integer.valueOf(127) == Integer.valueOf(127) zwróci true?**
18. **Czy Integer.valueOf(128) ==Integer.valueOf(128) zwróci true?**

Autor: Jakub Płonka

Zadanie - lista oparta o referencje - double linked list



1. Zmień implementację `StringListElement`, tak aby obsługiwała przechodzenie w obie strony
2. Spraw, aby `StringListElement` implementowało `ListIterator`, a do `StringList` dodaj metodę `listIterator()` zwracającą iterator
3. Poproś użytkownika o podanie elementów listy po spacji

np. “ala ma dużego kota” - 4 elementy, wypisz listę zgodnie z kolejnością wstawienia i odwrotnie np. “kota dużego ma ala”

StringListElement
- value: String - nextElement: ListElement - prevElement: ListElement
+ next(): ListElement + prev(): ListElement + setNext(ListElement) + setPrevious(ListElement) + getValue(): String + setValue(String)

StringList
- first: ListElement - last: ListElement - ???
+ head(): String + tail(): String + add(String) + size(): long + remove(String) + get(long): String

Autor: Jakub Płonka

Złożoność obliczeniowa

1. ***Czasowa złożoność obliczeniowa*** *- przyjętą miarą złożoności czasowej jest liczba operacji podstawowych w zależności od rozmiaru wejścia. Pomiar rzeczywistego czasu zegarowego jest mało użyteczny ze względu na silną zależność od sposobu realizacji algorytmu, użytego kompilatora oraz maszyny, na której algorytm wykonujemy. Dlatego w charakterze czasu wykonania rozpatruje się zwykle liczbę operacji podstawowych (dominujących). Operacjami podstawowymi mogą być na przykład: podstawienie, porównanie lub prosta operacja arytmetyczna.*

https://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa, dostęp 11.11.2018

2. ***Pamięciowa złożoność obliczeniowa*** *- Podobnie jak złożoność czasowa jest miarą czasu działania algorytmu, tak złożoność pamięciowa jest miarą ilości wykorzystanej pamięci. Jako tę ilość najczęściej przyjmuje się użytą pamięć maszyny abstrakcyjnej (na przykład liczbę komórek pamięci maszyny RAM) w funkcji rozmiaru wejścia. Możliwe jest również obliczanie rozmiaru potrzebnej pamięci fizycznej wyrażonej w bitach lub bajtach.*

https://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa, dostęp 11.11.2018

Złożoność obliczeniowa

1. Złożoność obliczeniowa sumowania - zakładamy, że operacją dominującą jest przypisanie

```
public class Main {

    public static void main(String[] args) {
        List<Integer> tab = Arrays.asList(1,2,3,4,5);
        Integer sum = 0;
        for(Integer i : tab)  {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Jak to wygląda na wykresie?

Notacja dużego O

1. Górna granica złożoności obliczeniowej
2. Stałe pomijamy (przy odpowiednio dużych wartościach będą nieznaczące)
3. Przykładowe rzędy złożoności funkcji posortowane rosnąco (zobaczmy na wykresach):
   - *O(1)* stała,
   - *O(logn)* logarytmiczna,
   - *O(n)* liniowa,
   - *O(nlogn)* liniowo-logarytmiczna,
   - *O(n^2)* kwadratowa,
   - *O(n^k)*, *k* jest stałą - wielomianowa,
   - *O(k^n)*, *k* jest stałą - wykładnicza,
   - *O(n!)* - rzędu silnia,

```
public class Main {

    public static void main(String[] args) {
        List<Integer> tab = Arrays.asList(1,2,3,4,5);
        Integer sum = 0;
        for(Integer i : tab)  {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Złożoność obliczeniowa operacji na LinkedList

1. Wstawianie
2. Wstawianie w miejscu n
3. Usuwanie
4. Pobieranie ntego element

Autor: Jakub Płonka

Sortowanie bąbelkowe

```
procedure bubbleSort( A : lista elementów do posortowania )
  n = liczba_elementów(A)
   do
    for (i = 0; i < n-1; i++) do:
      if A[i] > A[i+1] then
        swap(A[i], A[i+1])
      end if
    end for
    n = n-1
  while n > 1
end procedure
```

https://pl.wikipedia.org/wiki/Sortowanie_b%C4%85belkowe

Przerywnik - pytania na rozmowę rekrutacyjną

1. **Czym są magic strings, czy powinniśmy je zostawiać w kodzie?**
2. Czym jest maven? Czy są jakieś alternatywy dla niego?
3. Co to code review?
4. Czy “a”.equals(new String(“a”)) zwróci true?
5. Czym jest POJO?
6. Jak posortować Listę?
7. Jakie łańcuchy znaków dopasują się do wyrażenia regularnego: ala* ma kota+
8. Jak odczytać plik
9. Po co jest StringBuilder, StringBuffer?
10. **Jaki jest kontrakt pomiędzy equals, a hashCode?**
11. ***Jak działa HashMapa?**

Autor: Jakub Płonka

Sortowanie bąbelkowe - bubble sort - zadanie

1. Za pomocą sortowania bąbelkowego posortuj liczby wprowadzone po spacji przez użytkownika (Long):
   - wypisuj jak w poszczególnych iteracja wygląda lista
   - zwizualizuj wygląd listy za pomocą słupków w poszczególnych krokach, np: 3,2,1,4:

   ***

   **

   *

   ****

   - po zakończeniu sortowania wypisz ile operacji dominujących (porównań) zostało wykonanych na liście
2. Co jeśli zostanie podany posortowany ciąg znaków? Sprawdź i zastanów się nad optymalizacją
3. Jaka jest złożoność obliczeniowa
4. *Zmień typ do posortowania na BigDecimal

Autor: Jakub Płonka

Wyszukiwanie - zadanie

1. Znajdź w liście elementów {1,3,4,8,33,2,77} na której pozycji znajduje się 77, oblicz złożoność obliczeniową O

Autor: Jakub Płonka

Wyszukiwanie liniowe

Autor: Jakub Płonka

Wyszukiwanie binarne - zbiór musi być uporządkowany

1. Wejście: szukany element
2. Przypisz zmiennej lewa pozycję pierwszego element tablicy
3. Przypisz zmiennej prawa pozycję ostatniego elementu tablicy
4. Znajdź środek pomiędzy lewą, a prawą (pamiętaj, że ilość elementów tablicy może być parzysta)
5. Sprawdź czy lewa jest mniejsza lub równa prawej, jeśli nie - nie znaleziono elementu. KONIEC
6. Sprawdź czy na pozycji środek znajduje się szukany element, jeśli tak szukany element znajduje się na pozycji środek. KONIEC
7. Sprawdź czy element na pozycji środek jest większy od szukanego, jeśli tak ustaw prawa na środek - 1, jeśli nie:
8. Sprawdź czy element na pozycji środek jest mniejszy od szukanego, jeśli tak ustaw lewa na środek + 1.
9. Przejdź do punktu 5

10. Zaimplementuj i sprawdź czy działa, jaka jest złożoność obliczeniowa

Autor: Jakub Płonka

Wyszukiwanie binarne - zbiór musi być uporządkowany

1. Wejście: szukany element
2. Przypisz zmiennej lewa pozycję pierwszego element tablicy
3. Przypisz zmiennej prawa pozycję ostatniego elementu tablicy
4. Znajdź środek pomiędzy lewą, a prawą (pamiętaj, że ilość elementów tablicy może być parzysta)
5. Sprawdź czy lewa jest mniejsza lub równa prawej, jeśli nie - nie znaleziono elementu. KONIEC
6. Sprawdź czy na pozycji środek znajduje się szukany element, jeśli tak szukany element znajduje się na pozycji środek. KONIEC
7. Sprawdź czy element na pozycji środek jest większy od szukanego, jeśli tak ustaw prawa na środek - 1, jeśli nie:
8. Sprawdź czy element na pozycji środek jest mniejszy od szukanego, jeśli tak ustaw lewa na środek + 1.
9. Przejdź do punktu 5

10. Zaimplementuj i sprawdź czy działa, jaka jest złożoność obliczeniowa

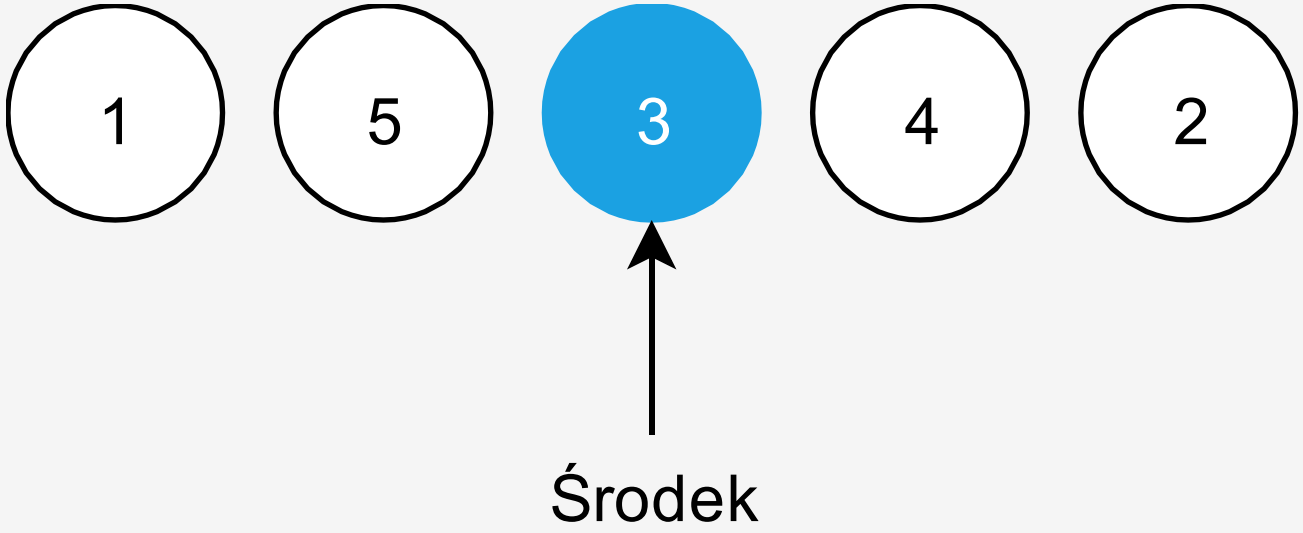
Autor: Jakub Płonka

Sortowanie przez zliczanie (counting sort)

1. Ograniczenie - musimy znać jaki jest maksymalny element
2. Wejście: {6, 3, 7, 1}
3. Maksymalny element to 7
4. Zliczamy ile razy poszczególne elementy występują w tablicy:
   - 6 występuje raz
   - 3 występuje raz
   - 7 występuje raz
   - 1 występuje raz
5. Informację o tym ile razy występuje dana liczba zapisujemy do tablicy pomocniczej o wielkości równej wartości maksymalnego elementu (7), gdzie indeksem będzie liczba, a wartością ilość wystąpień:
   - tab[6] = 1
   - tab[3] = 1
   - tab[7] = 1
   - tab[1] = 1
6. Przechodzimy przez tablicę tab pętlą i jeśli wartość podanego elementu jest większa od zera (czyli jego ilość wystąpień jest większa od zera) wypisujemy go i dekrementujemy ten element w tab.
   1. tab[0] = 0 - nic nie robię
   2. tab[1] = 1 - wypisuję i odejmuję jeden od tab[1], teraz równa się 0, nie wypisuję dalej
   3. tab[2] = 0 - nic nie robię
   4. tab[3] = 1 - wypisuję i odejmuję jeden od tab[3], teraz równa się 0, nie wypisuję dalej
   5. tab[4] = 0 - nic nie robię
   6. tab[5] = 0 - nic nie robię
   7. tab[6] = 1 - wypisuję i odejmuję jeden od tab[1], teraz równa się 0, nie wypisuję dalej
   8. tab[7] = 1 - wypisuję i odejmuję jeden od tab[1], teraz równa się 0, nie wypisuję dalej

Autor: Jakub Płonka

Sortowanie szybkie(quick sort)

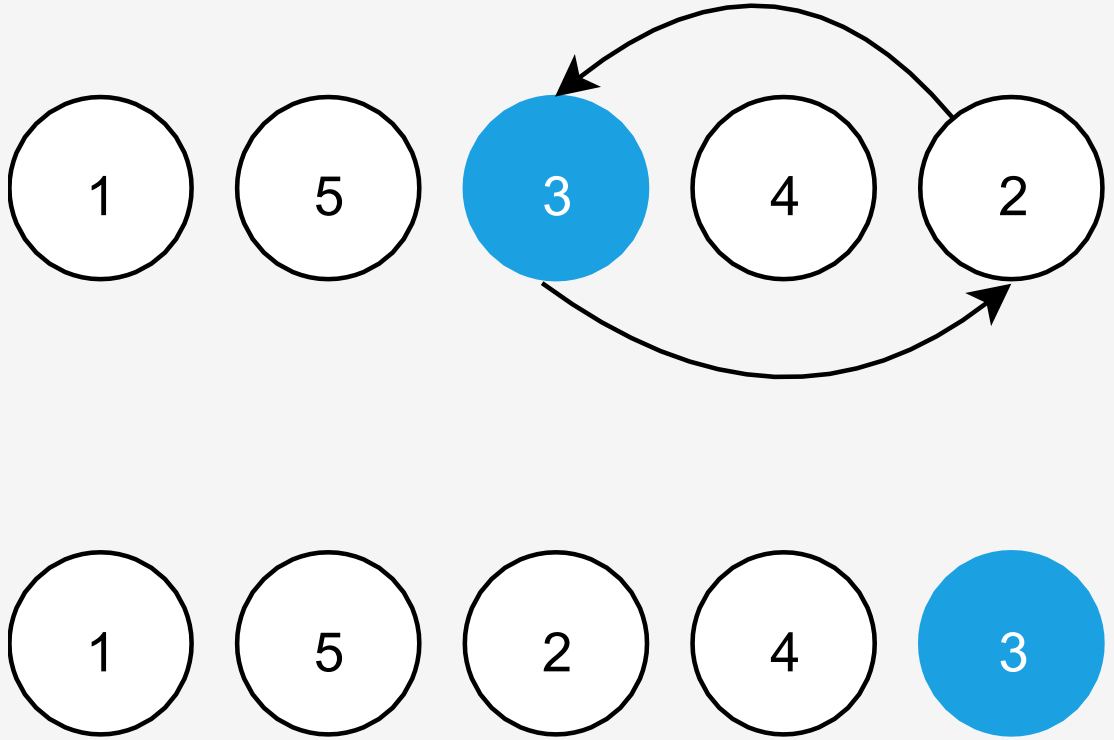


1. Wejście: {1,5,3,4,2}
2. Chciałbym, żeby po lewej od 3 (od środka) były element mniejsze od 3 (środka), a po prawej pozostałe

Autor: Jakub Płonka

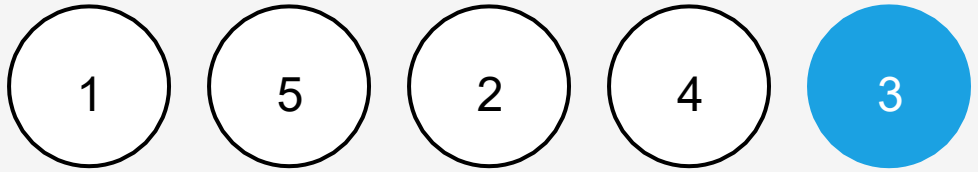
Sortowanie szybkie(quick sort)

Zamieniam miejscami środek z ostatnim elementem



Sortowanie szybkie(quick sort)

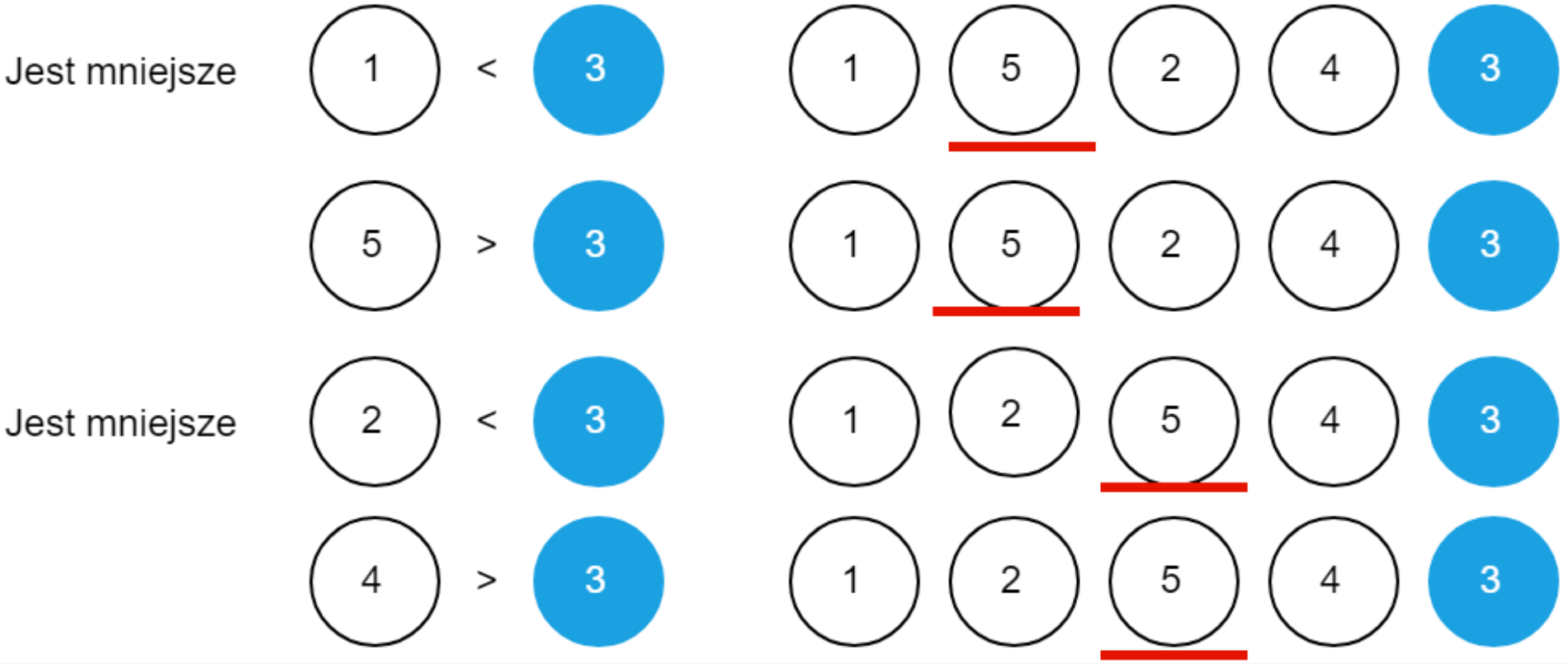
Przesuwam elementy tak, żeby mniejsze od środka (3) były po lewej, a pozostałe po prawej elementu środkowego



Autor: Jakub Płonka

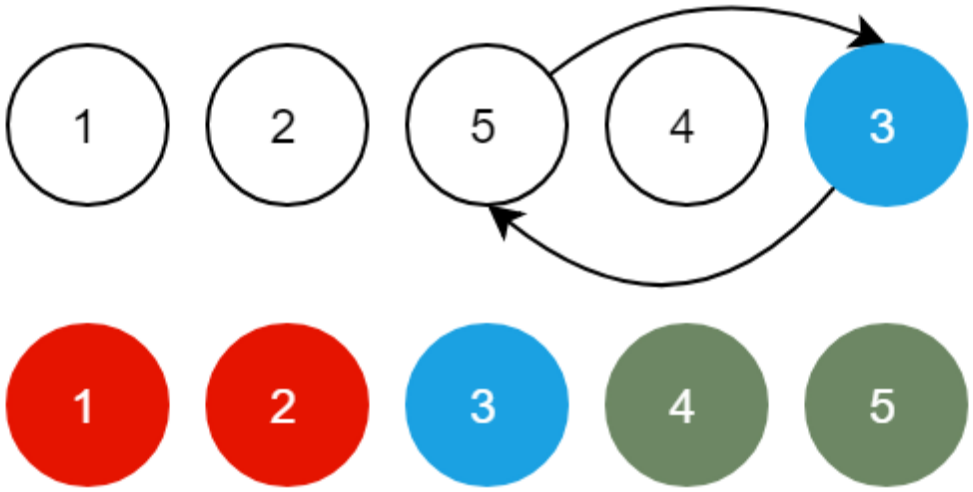
Sortowanie szybkie(quick sort)

Sprawdzam czy kolejne elementy są mniejsze od środka, jeśli tak zamieniam wstawiam je (a właściwie zamieniam) po ostatnio zmienionej pozycji (jeśli jeszcze nie ma żadnej to zaczynam od pozycji zero) (podkreślona na czerwono)



Sortowanie szybkie(quick sort)

Zamieniam miejscami element, który został wyznaczony jako środkowy z pozycją, która ostatnio została zmieniona + 1



W ten sposób uzyskałem tablicę podzieloną na 2 części:
elementy mniejsze od 3 (czerwone) i elementy większe lub równe 3 (zielone).

Na obu częściach powtarzam wszystkie operacje

Lista oparta o tablicę

1. Zaimplementuj listę opartą o tablicę, powinna mieć metody:
   - `get(int index)`
   - `add(String element)`
   - `add(int index, String element)`
   - `remove(int index)`
   - `size()`
   - `*listIterator`
2. Dodaj do niej elementy: 1,2,3,4,5
3. Usuń trzeci element
4. Wypisz wszystkie elementy
5. *Zaimplementuj interfejs `Iterable` oraz stwórz iterator implementujący `Iterator`, `ListIterator`, co jeśli w trakcie iterowania ktoś usunie element?

Przydatne będą metody:

- `Array.copyOf`
- `System.arraycopy`

Autor: Jakub Płonka

Kiedy czego użyć LinkedList vs ArrayList

- Wczytuję dane z pliku, a następnie wyświetlam je użytkownikowi
- W aplikacji mam zmienną typu List, użytkownik może do niej usuwać i dodawać element
- Okazuje się, w wyżej wymienionej aplikacji można też wyświetlać element
- Jeśli nie wiesz czego użyć w danym momencie, użyj..

Autor: Jakub Płonka

Złożoność obliczeniowa operacji na ArrayList

1. Wstawianie
2. Wstawianie w miejscu n
3. Usuwanie
4. Pobieranie ntego element

Autor: Jakub Płonka

Testy wydajnościowe ArrayList vs LinkedList, duży wolumen

1. Wstawianie
2. Wstawianie w miejscu n
3. Usuwanie
4. Pobieranie ntego element

Autor: Jakub Płonka

Jak działa HashMap

Kolejka - FIFO

- *First In, First Out; pierwszy na wejściu, pierwszy na wyjściu*
- *ludzie stojący w kolejce*
- *offer(e)*
- *poll()*
- *peek()*

Autor: Jakub Płonka

Stos - LIFO

- *Last In, First Out; ostatni na wejściu, pierwszy na wyjściu*
- *Stos kart do gry*
- *Stos kart wyborczych w urnie (najpewniej liczenie zacznie się od tych najbardziej na wierzchu)*
- *Stos wywołań funkcji*
- *push(e)*
- *pop()*
- *peek()*

Autor: Jakub Płonka

Co jest czym

Ludzie stojący do lekarza

Woda w rurach

Kolejność odkładania i mycia naczyń

Konstrukcja i dekonstrukcja piramidy Cheopsa

Slajdy w prezentacji

Ubrania w szafie

Kolejka - zadanie

Zaimplementuj interfejs java.util.Queue za pomocą wskazań na poszczególne elementy (tak samo jak w przypadku single linked list)

Autor: Jakub Płonka

Stos - zadanie

Zaimplementuj interfejs java.util.Stack za pomocą tablicy (tak samo jak w przypadku array list)

Autor: Jakub Płonka

Kolejka priorytetowa - PriorityQueue

Autor: Jakub Płonka

Kolejka priorytetowa - zadanie

Twoim zadaniem jest napisanie aplikacji służącej do decydowania o kolejności przyjęć w szpitalu.
Aplikacja powinna pozwalać na:

1. Rejestrację nowego pacjenta
2. Wyświetlanie aktualnego stanu kolejki (kto w niej stoi, nieposortowane, jaka jest długa)
3. Pobieranie następnej osoby z kolejki
4. Podglądanie kto jest następny w kolejce
5. Posiadać tryb demo w którym:
   a) zamiast ręcznie dodawać osoby aplikacja będzie startowała z 10 osobami w kolejce
   b) Co 2 sekundy aplikacja dodaje losową osobę
   c) Co 2 sekundy + random max 1 aplikacja przyjmuje pacjenta
6. *Napisz testy do aplikacji

1. Utwórz klasę Patient mającą pola imie (String), nazwisko (String), jakBardzoZly (int), rozpoznanaChoroba (enum z polem zaraźliwosc - GRYPA(1), PRZEZIEBIENIE(2), BIEGUNKA(3), COS_POWAZNEGO(4)) - pamietaj o getterach, setterach (chyba, że stwórzysz to jako immutable) Java Style i angielskich nazwach
2. Utwórz klasę HospitalQueueService mającą pole typu PriorityQueue, utwórz metodę add(Patient) oraz Patient next() -> pobierającą kolejną osobę z kolejki, Patient peek() -> podglądające kto jest następny()
   a) Metoda next powinna zwracać najpierw osoby o nazwisku “Kowalski” (to nazwisko ordynatora), w następnej kolejności powinna zwracać osoby z czymś poważnym, dalej osoby, których iloczyn jakBardzoZly i zaraźliwość będzie wyższy - zerknij na interfejs `Comparable` i konstruktor `PriorityQueue`
3. W menu, które stworzysz w Main powinny być trzy pozycje:
   a) Następny - wywołujące next i wypisujące kto jest następny (i zdejmujące tą osobę z kolejki)
   b) Kto następny - wywołujące peek()
   c) Nowy pacjent - umożliwiające podanie imienia, nazwiska, złości i rozpoznanej choroby, a następnie wrzucające to na kolejkę
4. Tryb demo - dodaj klasę PatientGenerator z metodą get z której będziesz zwracać osobę z losowym imieniem, nazwiskiem, stamen złości i rozpoznaną chorobą
   a) Imiona i nazwiska losuj z list, w których zahardkodowaleś przykładowe imiona i nazwiska
   b) *Imiona i nazwiska ładuj z pliku zamiast hardkodować
5. *Nazwisko ordynatora zamiast hardkodować wczytaj z pliku properties
6. *Nawet nie nazwisko, a listę nazwisk, ordynatorów może być wielu - mają pierwszeństwo ☺

Autor: Jakub Płonka

Labirynt

- Z pliku wczytaj do tablicy dwuwymiarowej labirynt, jest zapisany w postaci * i spacji - * to ściana, spacja wolna przestrzeń (dam Wam plik)
- https://gluonhq.com/products/scene-builder/#download
- Zaimplementuj logikę konieczną do poruszania się po labiryncie “ręcznie”
- Metoda lewej ręki