

Cześć!

Przemysław Grzesiowski

Powtórka

- Co to jest klasa?
- Co to jest obiekt?
- Co oznacza słowo `this`?
- Co oznacza słowo `static` w zależności od tego w którym miejscu występuje?
- Przy pomocy jakiej metody porównujemy obiekty?
- Jakiej klasy zazwyczaj używamy do precyzyjnego przechowywania liczb (np. salda rachunku)?
- Do czego służą pakiety?

Powtórka

- Czym jest hermetyzacja?
- Do czego służą gettery i settery? Po co ich używamy?
- Czym jest enum?
- Czym się różnią klasy immutable i mutable?
- Do czego służy rzutowanie? Co się stanie jak użyjemy go w niewłaściwy sposób?
- Czym jest String Pool?
- Jakie mamy instrukcje sterujące?
- Jakie mamy pętle w Javie?

Agenda

- 1.** Wstęp
- 2.** Rozwinięcie
- 3.** Zakończenie

Agenda

- 1.** Czym są struktury danych
- 2.** Czym jest przetwarzanie danych
- 3.** Tablice
- 4.** Typy generyczne
- 5.** Kolekcje
- 6.** Listy
- 7.** Komparatory

Agenda

- 9.** Iteratory
- 10.** Sety
- 11.** Mapy
- 12.** Kolejki
- 13.** Czas - daty
- 14.** Properties
- 15.** Locale

Struktura danych

Struktura danych

Struktura danych - sposób przechowywania danych w pamięci komputera

Przykładowo: rekord, tablica, lista itp.

Przetwarzanie danych

Przetwarzanie danych

Przetwarzanie danych - przetwarzanie danych wejściowych poprzez wykonywanie różnych operacji w celu otrzymania wyniku

Algorytm

Algorytm

Algorytm - skończony ciąg czynności koniecznych do rozwiązania problemu (ogólny opis jak rozwiązać problem). Algorytmy można implementować w programach.

Prosty przykład algorytmu to przepis kulinarny.

[Tablica] (Array)



Jak wyobrażasz sobie tablicę?

narysuj ...

Array is like a drawer ...

Gdzie są moje skarpetki?



Tablica

- struktura danych gromadząca uporządkowane dane

Tablica

- struktura danych gromadząca uporządkowane dane
- można tworzyć jednowymiarowe lub wielowymiarowe

Tablica

- struktura danych gromadząca uporządkowane dane
- można tworzyć jednowymiarowe lub wielowymiarowe
- mają stałą wielkość (określoną z góry podczas inicjacji)

Tablica

- struktura danych gromadząca uporządkowane dane
- można tworzyć jednowymiarowe lub wielowymiarowe
- mają stałą wielkość (określoną z góry podczas inicjacji)
- odwołujemy się do elementu po indeksie
numerujemy od 0 !!

```
int[] array = new int[10];      // wypełniona zerami
int[] array2 = {1, 2, 3, 4, 5}; // od razu zainicjalizowana danymi

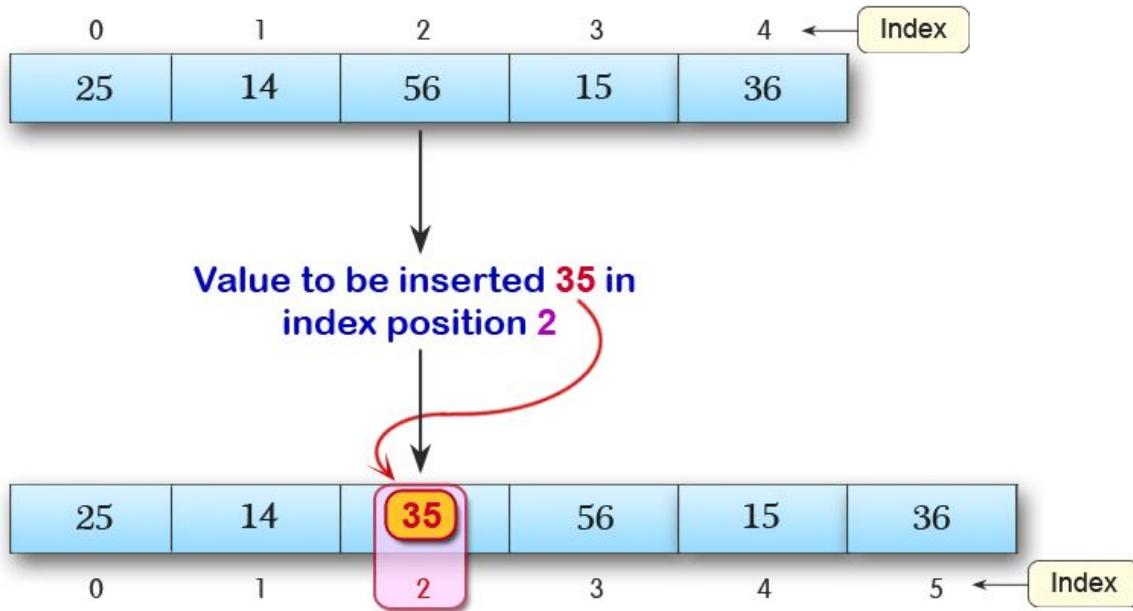
int number = array2[2];        // 3
```

Jak jest zorganizowana pamięć po inicjacji tablic?

```
String [ ] names = new String [5];
```

```
int [ ] numbers = new int[5];
```

Wstawiamy element do środka - ufff...



Pętla for each

Po tablicy można “przejść” używając pętli:

```
for (int i : array) {  
}
```

Klasa Arrays

Klasa `java.util.Arrays` ma różne przydatne metody do operowania na tablicach

Klasa Arrays

Klasa `java.util.Arrays` ma różne przydatne metody do operowania na tablicach, n.p.:

- `toString` - zamienia tablicę na czytelny ciąg znaków
- `copyOf` - kopiuje zawartość tablicy
- `sort` - sortuje zawartość tablicy
- `asList` - przekształca tablicę na listę
- `equals` - porównuje zawartość tablic

Zadanie 1



W klasie `ArrayTask`:

1. Stwórz 5-cio elementową tablicę intów i zainicjuj ją liczbami.
2. Wypisz zawartość tablicy na konsolę (używając pętli `for`)
3. Stwórz drugą tablicę o dwukrotnie większym rozmiarze, zawierającą elementy pierwszej tablicy (użyj klasy `Arrays`)
4. Wypisz zawartość drugiej tablicy (używając klasy `Arrays`)
5. Posortuj drugą tablicę
6. Wypisz zawartość posortowanej tablicy
7. (*) Spróbuj wyświetlić element 11-ty tablicy
8. (*) Sprawdź kod źródłowy klasy `Arrays`, spróbuj przeanalizować działanie wybranej metody (np. `sort`)
9. (*) napisz metodę dodającą nowy element na początek tablicy

Kiedy używamy tablic?

- z góry znamy rozmiar tablicy i wiemy, że się nie zmieni

Kiedy używamy tablic?

- z góry znamy rozmiar tablicy i wiemy, że się nie zmieni
- pracujemy ze zbiorami typów prostych (prymitywy) (np. wymuszona przez projekt, API bibliotek itp.)

Kiedy używamy tablic?

- z góry znamy rozmiar tablicy i wiemy, że się nie zmieni
- pracujemy ze zbiorami typów prostych (prymitywy) (np. wymuszona przez projekt, API bibliotek itp.)
- varargs

Kiedy używamy tablic?

- z góry znamy rozmiar tablicy i wiemy, że się nie zmieni
- pracujemy ze zbiorami typów prostych (prymitywy) (np. wymuszona przez projekt, API bibliotek itp.)
- varargs
- tworzymy kod krytyczny pod względem szybkości działania lub zajętości pamięci (temat zaawansowany)

Typy generyczne (ogólne)



```
public class ArrayListForIntegers {  
    // ...  
    public boolean add(Integer e) { ...}  
    // ...  
}
```

```
public class ArrayListForStrings {  
    // ...  
    public boolean add(String e) { ...}  
    // ...  
}
```

A może by tak uogólnić kod?

```
public class ArrayList<E> {  
    // ...  
  
    public boolean add(E e) { ... }  
    // ...  
}
```

Generic programming - definition

- writing code that can be reused for objects of many different types.

For example, you don't want to program separate classes to collect String and File objects. And you don't have to—the single class ArrayList collects objects of any class. This is one example of generic programming.

Programowanie generyczne

Pisanie kodu, który może być używany z różnymi typami.
Kompilator sprawdza poprawność typów.

Programowanie generyczne

Pisanie kodu, który może być używany z różnymi typami.
Kompilator sprawdza poprawność typów.

Przykładowo klasa `ArrayList` pozwala gromadzić obiekty typu `String`, `Integer`, `BigDecimal` i inne itp.

Programowanie generyczne

Pisanie kodu, który może być używany z różnymi typami.
Kompilator sprawdza poprawność typów.

Przykładowo klasa `ArrayList` pozwala gromadzić obiekty typu `String`, `Integer`, `BigDecimal` i inne itp.

```
List<String> strings = new ArrayList<>();
```

```
List<Integer> ints = new ArrayList<>();
```

Programowanie generyczne

Pisanie kodu, który może być używany z różnymi typami.
Kompilator sprawdza poprawność typów.

Przykładowo klasa `ArrayList` pozwala gromadzić obiekty typu `String`, `Integer`, `BigDecimal` i inne itp.

```
List<String> strings = new ArrayList<>();
```

```
List<Integer> ints = new ArrayList<>();
```

```
List<Object> objs = new ArrayList<>();
```

Klasy ogólne (parametryzowane)

Można tworzyć własne klasy parametryzowane:

```
public class MyClass<T> {
    private T field;

    public T getField() {
        return field;
    }
}
```

Klasy ogólne (parametryzowane)

Można również zdefiniować ograniczenia:

```
public class MyClass<T extends Number>
```

Klasy ogólne (parametryzowane)

Metoda ogólna w klasie, która nie jest parametryzowana:

```
public class MyClass {  
  
    public <T> T method(T parameter) {  
        // do something  
        return parameter;  
    }  
}
```

Zadanie 2



- 1.** Stwórz nową klasę Point (w pakiecie “generics”):
 - a.** z generycznym parametrem E ograniczonym do klas dziedziczących po Number
 - b.** z dwoma polami x, y o typie E oznaczającymi współrzędne punktu (o typie E)
 - c.** odpowiednim konstruktorem i getterami
- 2.** Przetestuj działanie klasy Point w klasie GenericsTask na przynajmniej dwóch różnych typach.
- 3.** (*) dodaj kolejne dwa typy numeryczne do testów

Kolekcje (Collections)



Kolekcje = ulepszone tablice

Collections and arrays are similar - they both hold references to **objects** and they can be managed as a group.

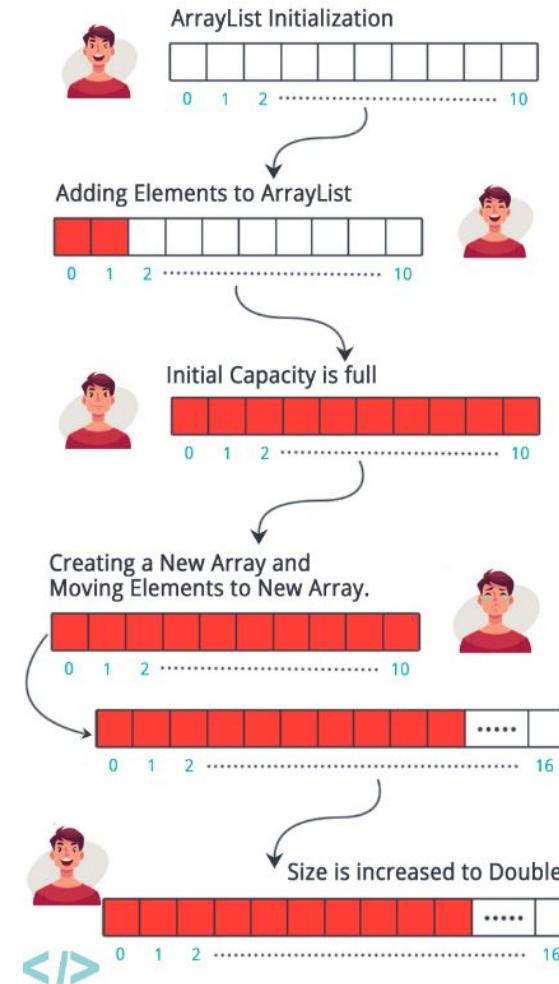
Kolekcje = ulepszone tablice

Collections (unlike arrays) do not need to be assigned a certain capacity when instantiated.

Kolekcje = ulepszone tablice

Collections (unlike arrays) do not need to be assigned a certain capacity when instantiated.

```
List<String> lista = new ArrayList<>();  
List<String> lista2 = new ArrayList<>( initialCapacity: 10 );  
  
String[] tablica = new String[10];
```



Kolekcje = ulepszone tablice

```
List<String> lista = new ArrayList<>();  
lista.add("Azor");  
System.out.println(lista.size());
```

```
String[] tablica = new String[10];  
tablica[0] = "Azor";  
System.out.println(tablica.length);
```

Kolekcje = ulepszone tablice

```
List<String> lista = new ArrayList<>();  
lista.add("Azor");  
System.out.println(lista.size());
```

```
String[] tablica = new String[10];  
tablica[0] = "Azor";  
System.out.println(tablica.length);
```

Collections can grow and shrink in size automatically when objects are added or removed.

Kolekcje = ulepszone tablice

```
List<int> lista = new ArrayList<int>();
```

Type argument cannot be of primitive type

Kolekcje = ulepszone tablice

```
List<int> lista = new ArrayList<int>();
```

Type argument cannot be of primitive type

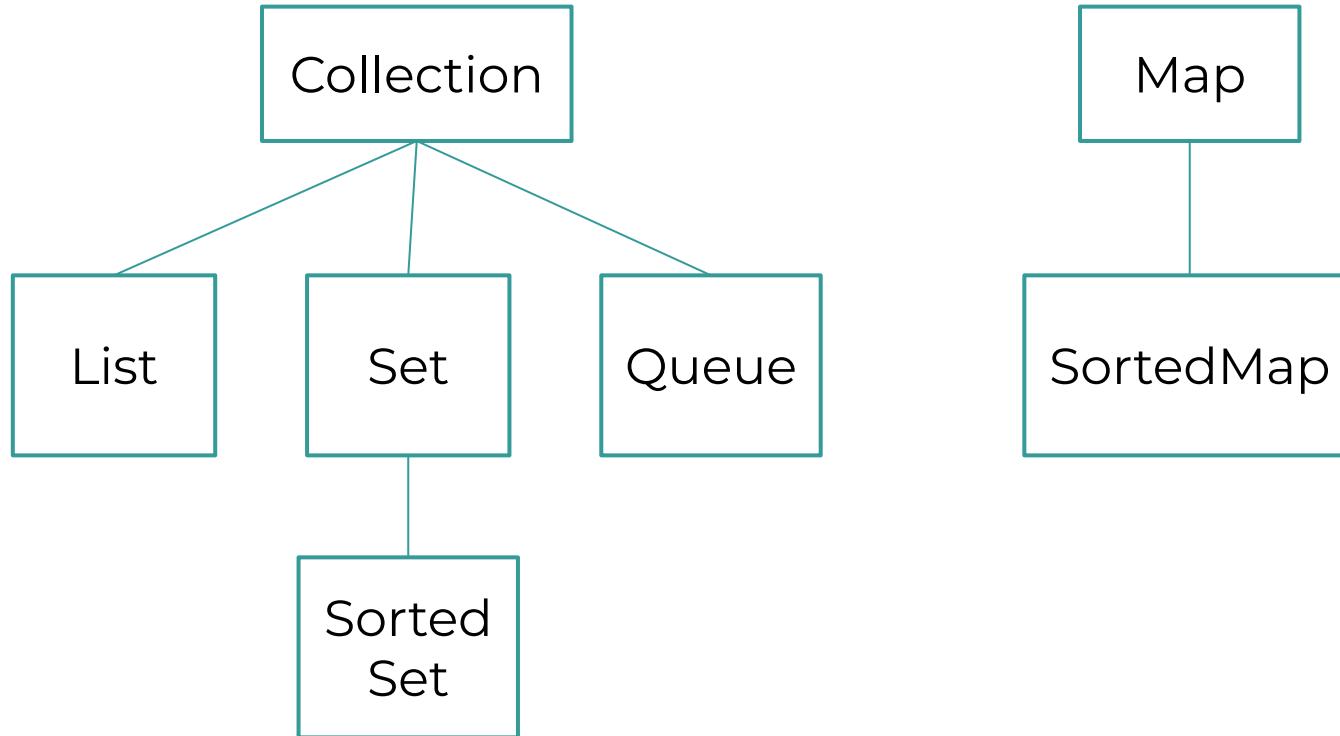
Collections cannot hold basic data type elements (primitive types) such as int, long, or double; instead, they hold Wrapper Classes such as Integer, Long, or Double

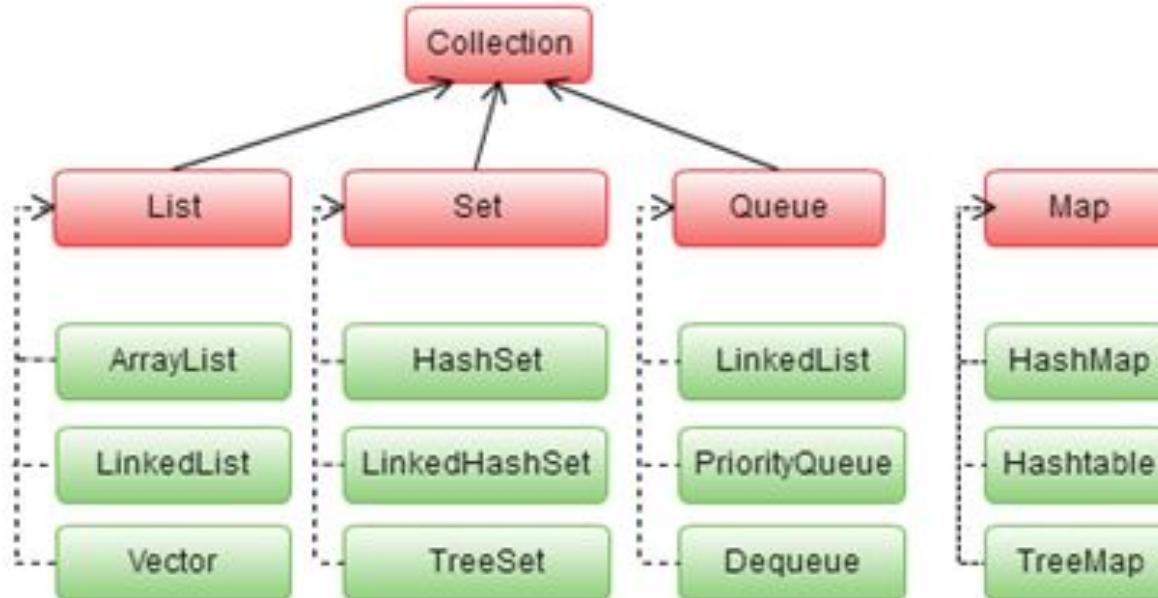
Collections Framework

- interfejsy
- implementacje
- algorytmy

http://files.zeroturnaround.com/pdf/zt_java_collections_cheat_sheet.pdf

Interfejsy kolekcji





Interfejs Collection

- add - dodaje element do kolekcji
- addAll - dodaje wszystkie elementy kolekcji
- clear - usuwa zawartość kolekcji
- contains - sprawdza czy kolekcja zawiera dany element
- isEmpty - sprawdza czy kolekcja jest pusta
- iterator - zwraca obiekt umożliwiający przechodzenie po kolekcji
- remove - usuwa dany element kolekcji (jeśli istnieje)
- size - zwraca rozmiar kolekcji
- toArray - przekształca kolekcję w tablicę
- i inne...

Methods

Modifier and Type	Method and Description
boolean	add(E e) Ensures that this collection contains the specified element (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	clear() Removes all of the elements from this collection (optional operation).
boolean	contains(Object o) Returns true if this collection contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
boolean	equals(Object o) Compares the specified object with this collection for equality.
int	hashCode() Returns the hash code value for this collection.
boolean	isEmpty() Returns true if this collection contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this collection.
boolean	remove(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	size() Returns the number of elements in this collection.
Object[]	toArray() Returns an array containing all of the elements in this collection.
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

Klasa Collections

Klasa ze statycznymi, pomocniczymi metodami, np. :

- sort `(List<T> list)` - sortuje listę
- frequency - zlicza liczbę wystąpień elementu w kolekcji
- max `(Collection<? extends T> coll)` - zwraca największy element kolekcji
- min `(Collection<? extends T> coll)` - zwraca najmniejszy
- reverse `(List<?> list)` - odwraca kolejność listy
- shuffle `(List<?> list)` - losowo mieszka elementy listy
- swap - zamienia elementy listy o podanych indeksach
- i inne...

Lista (List)



Shopping List

1. Mleko

2. Płatki

3. Olej

4. Mąka

5. Cukier

6. Dżem

7. Kukurydza

8. Jabłka

9. Groszek

10. Kiełbasa

11. Sok

12. Pomidory

13. Czekolada

14. Lody

15. Lizaki

Shopping List

- [0] 1. Mleko
- [1] 2. Płatki
- [2] 3. Olej
- [3] 4. Mąka
5. Cukier
6. Dżem
7. Kukurydza
8. Jabłka
9. Groszek
10. Kiełbasa
11. Sok
12. Pomidory
13. Czekolada
14. Lody
15. Lizaki

[16]

Lista (Interface List<E>)

- zachowuje kolejność elementów

Lista (Interface List<E>)

- zachowuje kolejność elementów
- pozwala na duplikaty elementów

Lista (Interface List<E>)

- zachowuje kolejność elementów
- pozwala na duplikaty elementów
- do elementów można odwoływać się po indeksie `get(int index)`

Lista (Interface List<E>)

- zachowuje kolejność elementów
- pozwala na duplikaty elementów
- do elementów można odwoływać się po indeksie `get(int index)`

Przykładowe implementacje interfejsu List:

- ArrayList
- LinkedList
- Stack

ArrayList implements List<E>

“mleko”	“piwo”	“chleb”	“ser”	“woda”
---------	--------	---------	-------	--------

ArrayList = prawie jak Array ...

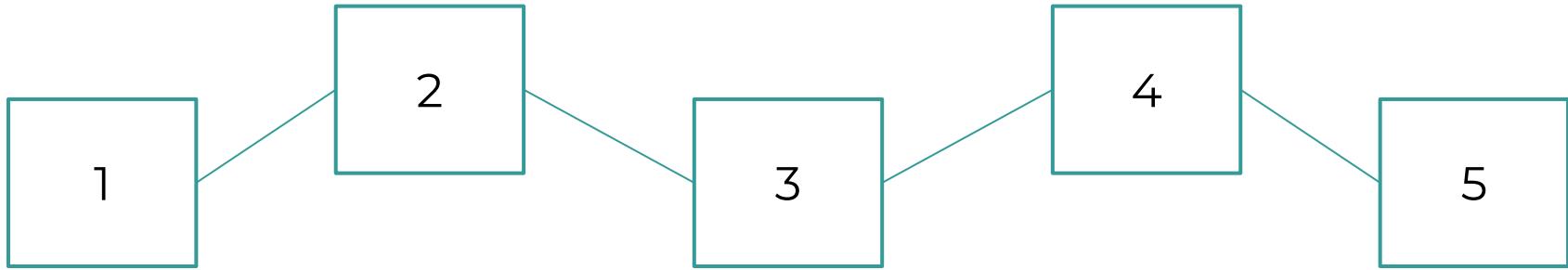
ArrayList implements List<E>

“mleko”	“piwo”	“chleb”	“ser”	“woda”
---------	--------	---------	-------	--------

ArrayList = prawie jak Array ... a może nawet lepiej?

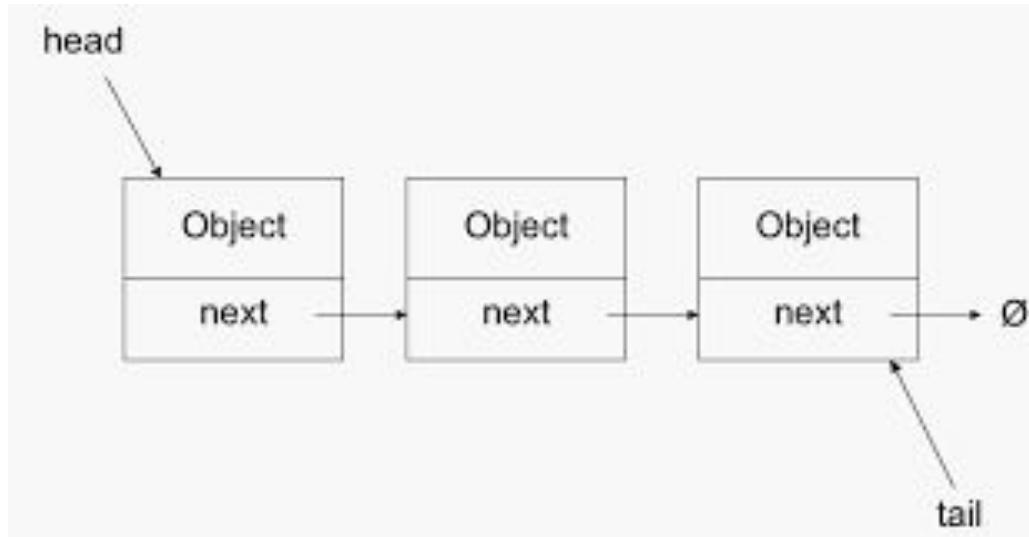
- szybki dostęp do elementu (jeśli znamy indeks)
- wolne usuwanie elementów z początku i środka
- łatwe dodawanie elementu na koniec

LinkedList implements List<E>



- szybkie dodawanie elementu na początku listy
- szybkie usuwanie elementu ze środka
- wolny dostęp do elementu o danym indeksie

LinkedList implements List<E>

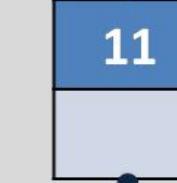
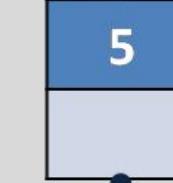


LinkedList<Integer> - przykład

START

Information

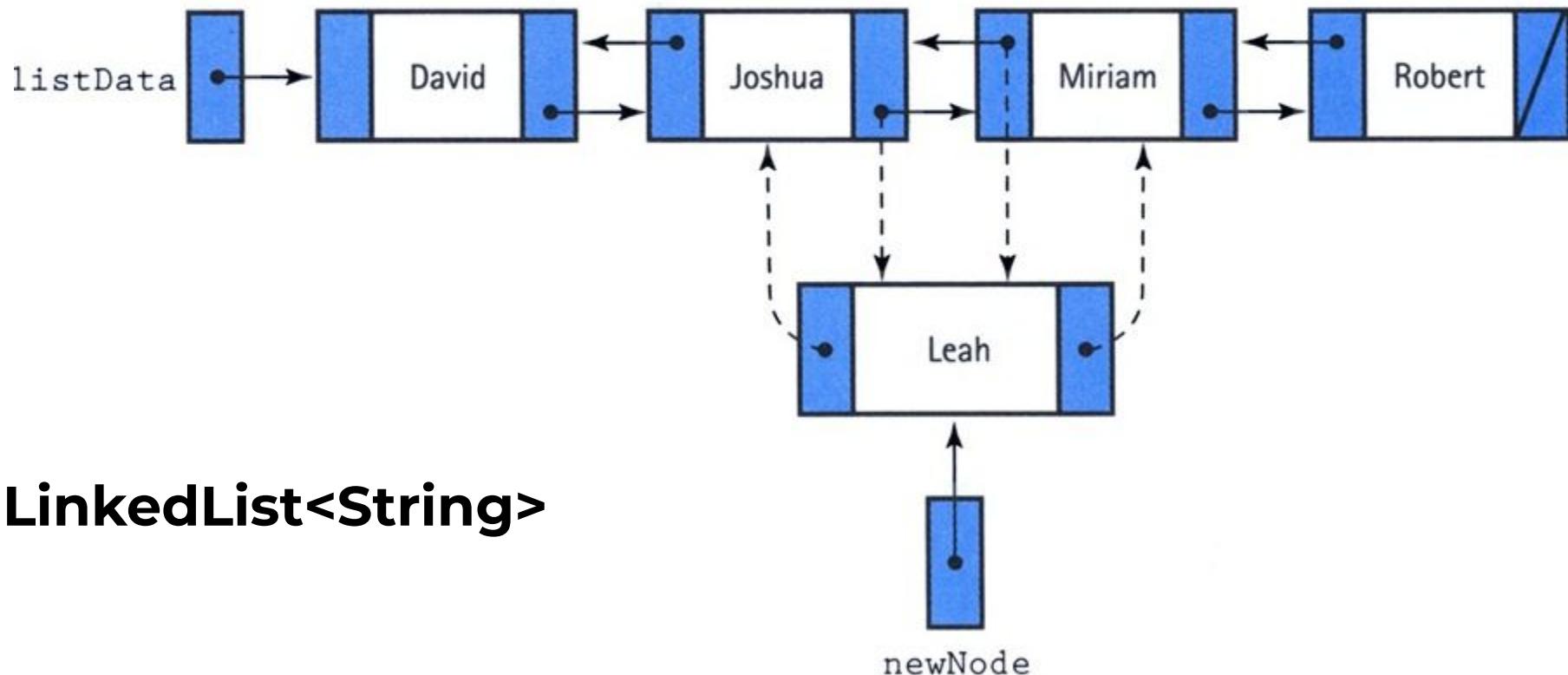
Next Pointer



Linked List

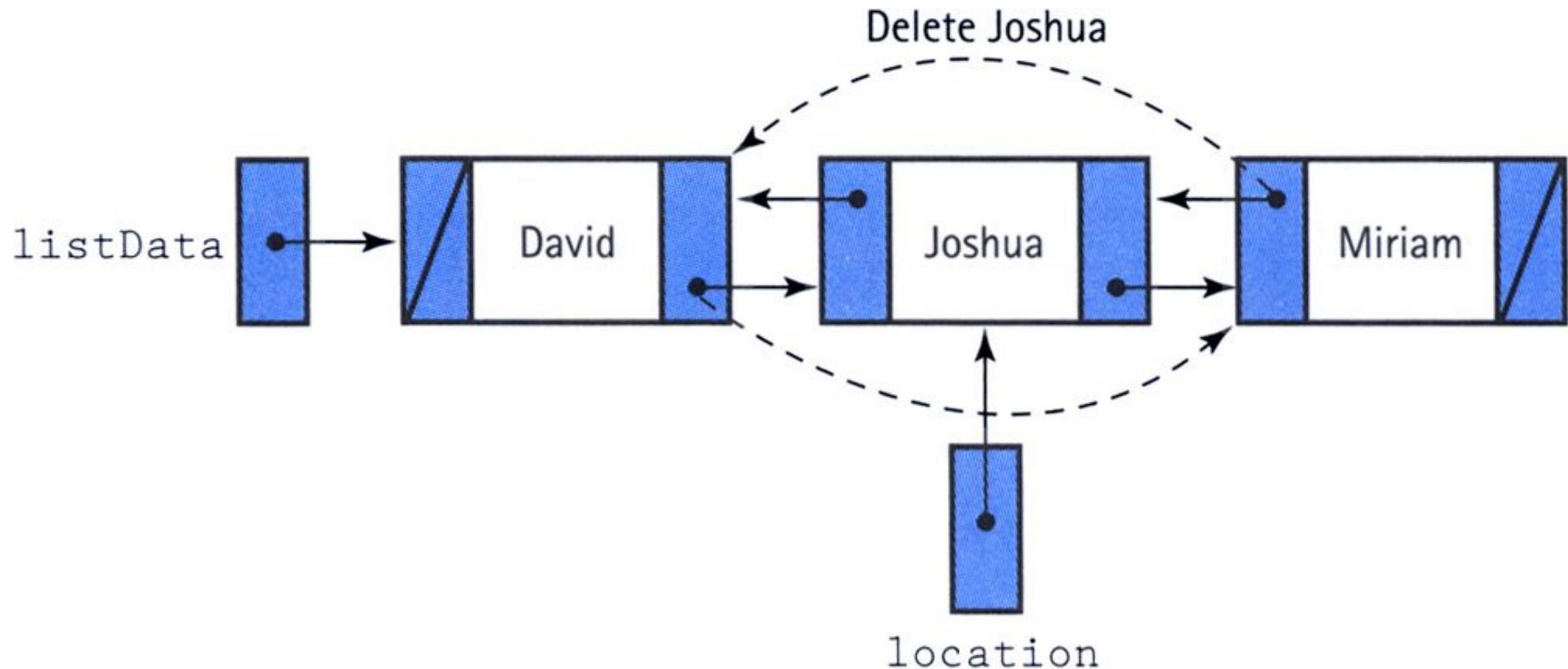
Crunchify

Inserting into a doubly linked list



LinkedList<String>

LinkedList<String>



wybrane metody interfejsu List

- `get(int index)` - zwraca element o danym indeksie
- `indexOf(Object o)` - zwraca indeks danego elementu
- `set(int index, E element)` - zamienia element o danym indeksie na podany w parametrze
- `toArray()` - zwraca tablicę utworzoną na podstawie listy
- `add(E e)` - dodaje element na koniec listy
- `add(int index, E element)` - wstawia element pod danym indeksem

Zadanie 3



- 1.** W pakiecie lists stwórz nową klasę Person zawierającą imię i nazwisko osoby.
- 2.** W klasie ListTask stwórz listę kilku osób (wykonaj zadanie na dwa sposoby - używając obu znanych Ci implementacji interfejsu List). Wrzuć stworzone listy na konsolę.
- 3.** Przetestuj działanie metod: get, indexOf, toArray, clear. Spróbuj coś “zepsuć” - np. otrzymać wyjątek.
- 4.** (*) Przećwicz działanie metod: set, addAll, remove, sublist.
- 5.** (*) ArrayList vs LinkedList - spróbuj napisać program który udowodni różnice w wydajności dodawania elementu do środka listy

ArrayList vs LinkedList



W jakim przypadku powinniśmy użyć ArrayList,
a w jakim LinkedList?

Jak posortować elementy listy?

Jak posortować elementy listy?

3,5,1

“A”, “a”, “b”

“Az”, “Ab”, “Abc”

Jak posortować elementy listy?

```
public class Person {  
    private String firstName;  
    private String lastName;  
    // ...  
}
```

Przemysław Grzesiowski
Adam Nowak





Comparable<T> - porządek “naturalny”
Comparator<T>

Dwa interfejsy -> dwie metody sortowania

<https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html>

```
static <T extends Comparable<? super T>>
void
static <T> void
```

<code>sort(List<T> list)</code>	Sorts the specified list into ascending order, according to the natural ordering of its elements.
<code>sort(List<T> list, Comparator<? super T> c)</code>	Sorts the specified list according to the order induced by the specified comparator.

Comparable<T>

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*.

```
int compareTo (T o)
```

Comparator<T>

Interfejs (z parametrem generycznym) służący do porównywania obiektów w kolekcjach.

Deklarując własny Comparator implementujemy metodę:

```
int compare (T o1, T o2)
```

Metoda compare porównuje dwa obiekty tego samego typu ze sobą

Zadanie 4



- 1.** Sprawdź działanie metody `compareTo` z klasy `String` na kilku wybranych stringach. Kiedy metoda zwraca 0?
- 2.** Stwórz `PersonSurnameComparator` sortujący osoby po nazwisku (porządek alfabetyczny)
- 3.** W klasie `ListTask` posortuj listę osób używając `Collections.sort`.
- 4.** (*) Wykonaj to samo sortowanie w porządku odwrotnym do alfabetycznego
- 5.** (*) Zaimplementuj interfejs `Comparable` w klasie `Person`, posortuj listę za jego pomocą.

Iterator<E>

Interfejs z parametrem generycznym służący do definiowania kolejności przetwarzania kolekcji

metody:

- next - zwraca kolejny element
- hasNext - zwraca true jeśli istnieje następny element
- remove - usuwa bieżący element (Removes the element returned by the most recent next or previous operation.)

Zadanie 5



W klasie ListTask spróbuj usunąć wszystkie osoby z listy używając pętli for each i zobacz co się stanie.

Zadanie 6



Zrealizuj poprzednie zadanie używając iteratora.

Zbiór (Set<E>)



Zbiór (Interface Set<E>)

- A collection that contains no duplicate elements.

Zbiór (Interface Set<E>)

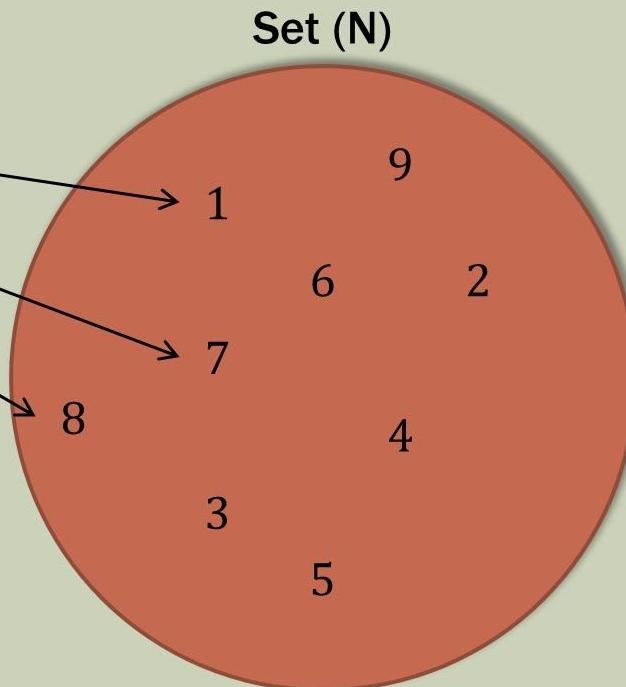
- A collection that contains no duplicate elements. More formally, sets contain no pair of elements `e1` and `e2` such that `e1.equals(e2)`, and at most one null element.

Zbiór (Interface Set<E>)

- A collection that contains no duplicate elements. More formally, sets contain no pair of elements `e1` and `e2` such that `e1.equals(e2)`, and at most one null element.
- As implied by its name, this interface models the mathematical set abstraction.

SETS AND ELEMENTS

Elements of the Set



“is an element of”

\in

Example: $4 \in N$

“is NOT an element of”

\notin

Example: $12 \notin N$

Zbiór (Interface Set<E>)

- elementy zbioru są unikalne
- zachowanie kolejności zależy od implementacji
- umożliwia szybkie sprawdzenie czy dany element znajduje się w zbiorze (metodą contains)

Zbiór (Interface Set<E>)

- elementy zbioru są unikalne
- zachowanie kolejności zależy od implementacji
- umożliwia szybkie sprawdzenie czy dany element znajduje się w zbiorze (metodą contains)

Przykładowe implementacje interfejsu Set:

- HashSet
- TreeSet
- LinkedHashMap

Methods

Modifier and Type	Method and Description
boolean	add(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	clear() Removes all of the elements from this set (optional operation).
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	equals(Object o) Compares the specified object with this set for equality.
int	hashCode() Returns the hash code value for this set.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set.
boolean	remove(Object o) Removes the specified element from this set if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	size() Returns the number of elements in this set (its cardinality).
Object[]	toArray() Returns an array containing all of the elements in this set.
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

HashSet

implements [Set<E>](#)

Nie gwarantuje kolejności.

Używa hash table.

HashSet

implements Set<E>

Nie gwarantuje kolejności.

Używa hash table.

```
Set<String> alkohole = new HashSet<>();
alkohole.add("wódka");
alkohole.add("spirytus");
alkohole.add("woda");
System.out.println(alkohole);
```

Zadanie 7



W klasie SetTask przetestuj działanie HashSet na obiektach typu Person.

Upewnij się, że zbiór prawidłowo działa w przypadku duplikatów.

TreeSet

implements Set<E>

```
Set<Integer> integers = new TreeSet<>();
integers.add(40);
integers.add(2);
integers.add(5);
System.out.println(integers);
```

Gwarantuje zachowanie kolejności (wg porządku
wyznaczonego przez implementacje interfejsu Comparable
lub przekazanego w konstruktorze komparatora).

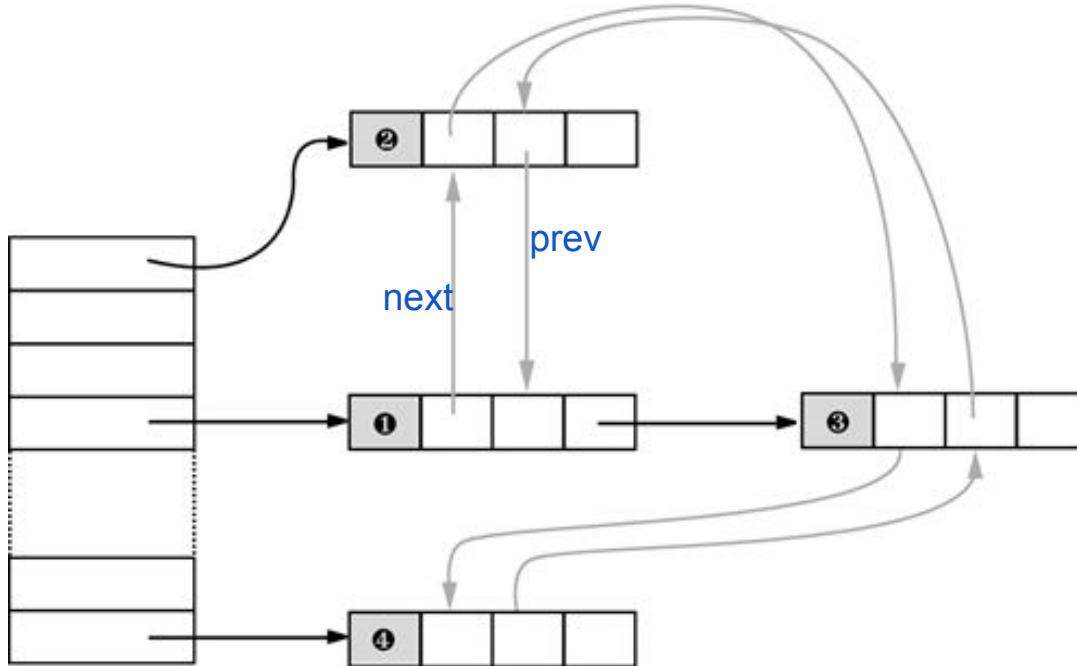
LinkedHashSet

implements [Set<E>](#)

Gwarantuje zachowanie kolejności wg dodawania elementów.

```
Set<Integer> integers = new LinkedHashSet<>();
integers.add(40);
integers.add(2);
integers.add(5);
System.out.println(integers);
```

LinkedHashTable



Zadanie 8



W klasie SetTask przetestuj działanie TreeSet i LinkedHashSet.

HashSet vs TreeSet vs LinkedHashSet



W jakim przypadku powinniśmy użyć HashSet,
TreeSet a w jakim LinkedHashSet?

Mapa (Map)



Map

An object that maps keys to values.

map (mæp)

n

1. (Surveying) a diagrammatic representation of the earth's surface or part of it, showing the geographical distributions, positions, etc, of natural or artificial features such as roads, towns, relief, rainfall, etc
2. (Astronomy) a diagrammatic representation of the distribution of stars or of the surface of a celestial body: *a lunar map*.
3. a maplike drawing of anything
4. (Mathematics) *maths* another name for function⁴
5. a slang word for *face*¹
6. **off the map** no longer important or in existence (esp in the phrase *wipe off the map*)
7. **put on the map** to make (a town, company, etc) well-known

vb (tr), maps, mapping or mapped

8. (Surveying) to make a map of
9. (Mathematics) *maths* to represent or transform (a function, figure, set, etc): the results were mapped onto a graph. See also *map out*

Array is like a drawer ...

Gdzie są moje skarpetki?



Map is like a drawer that stores things on bins and labels them

Gdzie są moje skarpetki?



Zamiast indeksów (0,1,2,...) używamy **kluczy** (key)

Map (Interface Map<K,V>)

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość (key-value pairs)

Map (Interface Map<K,V>)

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość (key-value pairs)
- do wartości odwołujemy się poprzez klucze
- klucze są unikalne (brak duplikatów) i są obiektami

Map (Interface Map<K,V>)

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość (key-value pairs)
- do wartości odwołujemy się poprzez klucze
- klucze są unikalne (brak duplikatów) i są obiektami
- some implementations prohibit null keys and values, and some have restrictions on the types of their keys.

Map (Interface Map<K,V>)

- struktura danych reprezentująca mapę (słownik)
- przechowuje pary klucz-wartość (key-value pairs)
- do wartości odwołujemy się poprzez klucze
- klucze są unikalne (brak duplikatów) i są obiektami
- some implementations prohibit null keys and values, and some have restrictions on the types of their keys.

- przykłady implementacji interfejsu Map:

[AbstractMap](#), [EnumMap](#), [HashMap](#), [Hashtable](#), [LinkedHashMap](#), [TreeMap](#), [WeakHashMap](#)

Methods

Modifier and Type	Method and Description
void	clear() Removes all of the mappings from this map (optional operation).
boolean	containsKey(Object key) Returns <code>true</code> if this map contains a mapping for the specified key.
boolean	containsValue(Object value) Returns <code>true</code> if this map maps one or more keys to the specified value.
<code>Set<Map.Entry<K, V>></code>	entrySet() Returns a <code>Set</code> view of the mappings contained in this map.
boolean	equals(Object o) Compares the specified object with this map for equality.
V	get(Object key) Returns the value to which the specified key is mapped, or <code>null</code> if this map contains no mapping for the key.
int	hashCode() Returns the hash code value for this map.
boolean	isEmpty() Returns <code>true</code> if this map contains no key-value mappings.
<code>Set<K></code>	keySet() Returns a <code>Set</code> view of the keys contained in this map.
V	put(K key, V value) Associates the specified value with the specified key in this map (optional operation).
void	putAll(Map<? extends K, ? extends V> m) Copies all of the mappings from the specified map to this map (optional operation).
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation).
int	size() Returns the number of key-value mappings in this map.
<code>Collection<V></code>	values() Returns a <code>Collection</code> view of the values contained in this map.

Map interface - wybrane metody

- containsKey - sprawdza czy w mapie jest element o danym kluczu (szybko działa)
- containsValue - sprawdza czy w mapie jest element o danej wartości (wolno działa)
- put - dodaje pary klucz-wartość do mapy
- get - zwraca wartość dla danego klucza
- keySet - zwraca zbiór kluczy
- entrySet - zwraca zbiór par klucz-wartość
- i inne...

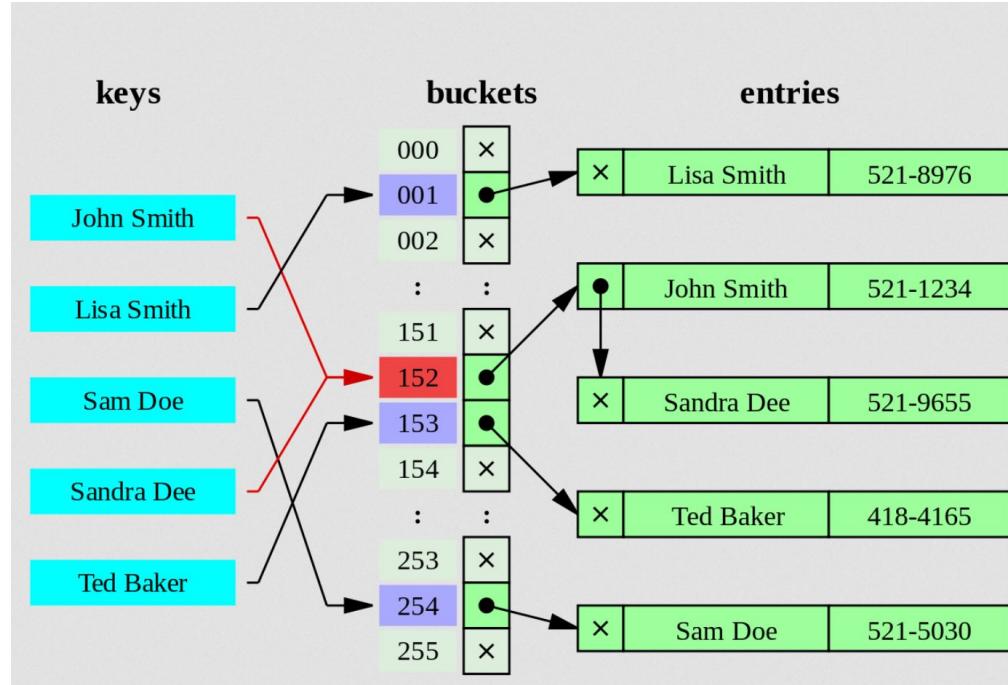
HashMap

 (implements [Map<K,V>](#))

- szybko wyszukuje obiekty po kluczu
- używa tablic hashujących

```
HashMap<Integer, String> hashMap = new HashMap<>();  
hashMap.put(1, "Jeden");  
hashMap.put(2, "Dwa");  
  
String text = hashMap.get(1); // Jeden
```

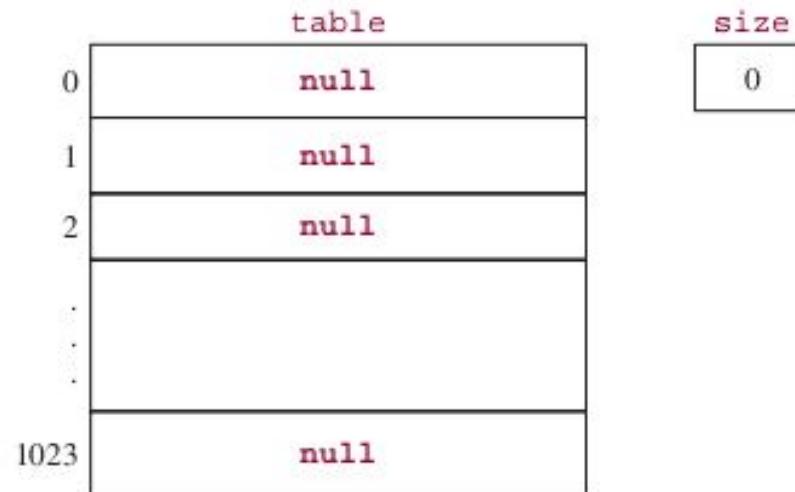
Tablice hashujące



By Jorge Stolfi - Own work, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=6471915>

Empty HashMap

Initial size = 1024



```
persons.put (251, "Smolenski");  
persons.put (118, "Schwartz");  
persons.put (335, "Beh-Forrest");
```

table	size
null	3
null	
null	
118	118
251	251
335	335
1023	null

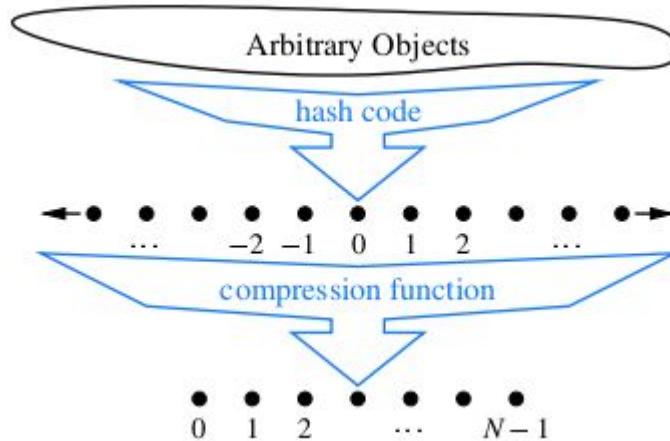
```
persons.put (214303261, "Albert");
persons.put (033518000, "Schranz");
```

Hashing -process of transforming
a key into an index in a table

```
new Integer(214303261).hashCode() % 1024      -> 541
new Integer(33518000).hashCode() % 1024        -> 432
```

table	size
0 null	2
1 null	
2 null	
432 033518000	
541 214303261	
1023 null	

Hash function = hash code + compression function



Two parts of a hash function:

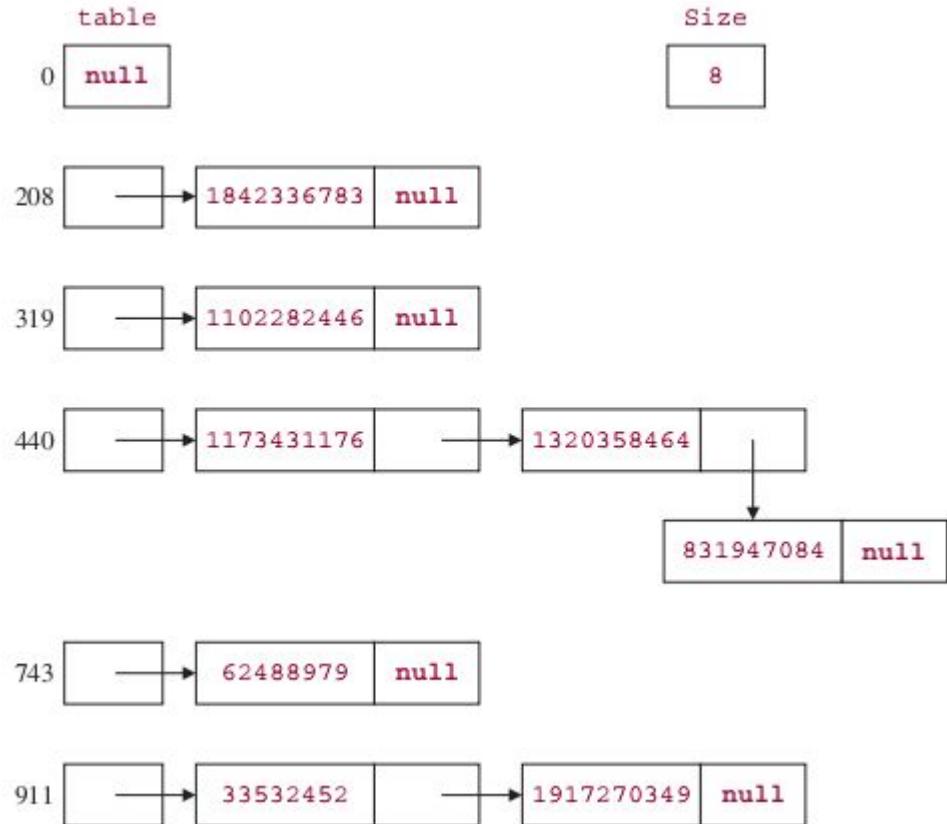
hash code - maps a key k to an integer,

compression function - maps the hash code to an integer within a range of indices, $[0, N - 1]$, for a bucket array

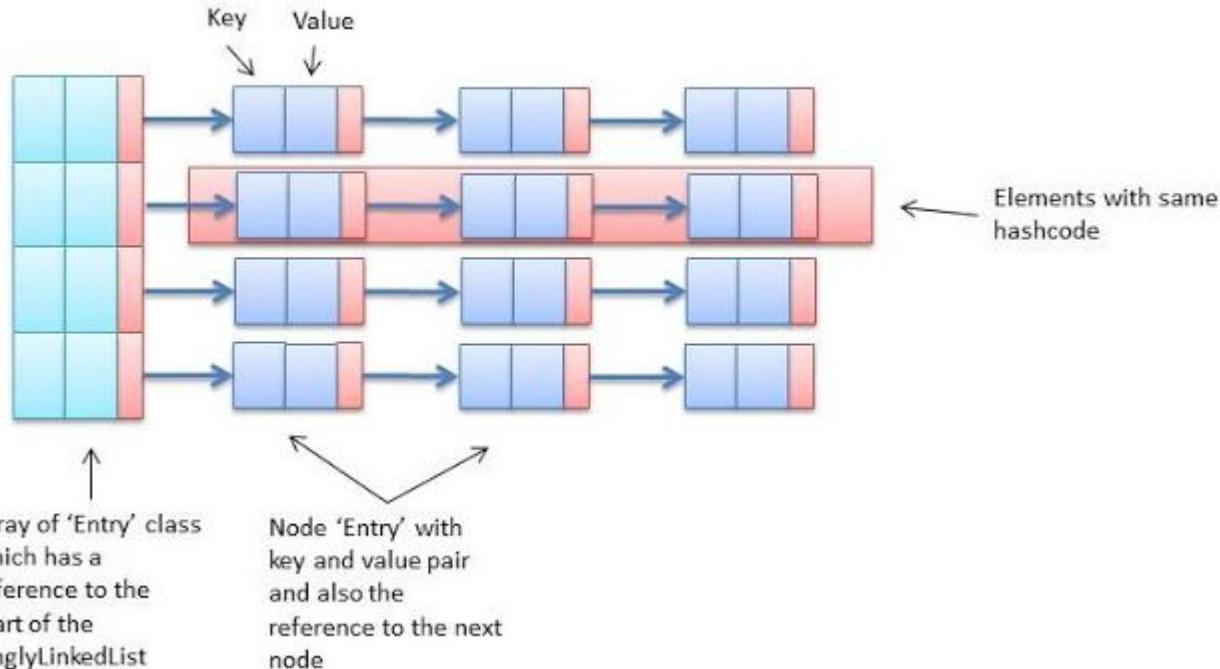
```
persons.put (1842336783, "Albert");
persons.put (1102282446, "Schranz");
```

results of hash function:

key	index
62488979	743
831947084	440
1917270349	911
1842336783	208
1320358464	440
1102282446	319
1173431176	440
33532452	911



HashMap structure



The default version of `hashCode()` provided by the `Object` class is often just an integer representation derived from the object's memory address.

We must be careful if relying on the default version of `hashCode()` when authoring a class. For hashing schemes to be reliable, it is imperative that any two objects that are viewed as “equal” to each other have the same hash code.

Zadanie 9



- 1.** W klasie `HashMapTask` dodaj do HashMapy kilka osób, załóż, że kluczem jest login. Dodaj przynajmniej dwie osoby o tym samym loginie.
- 2.** Pobierz jedną osobę używając jej loginu i wypisz jej dane.
- 3.** Użyj metody `entrySet` i iterując po kolekcji wypisz loginy oraz dane wszystkich osób.
- 4.** (*) Jaką metodę użyjesz do iterowania po samych wartościach (bez dostępu do kluczy)?

TreeMap (implements [Map<K,V>](#))

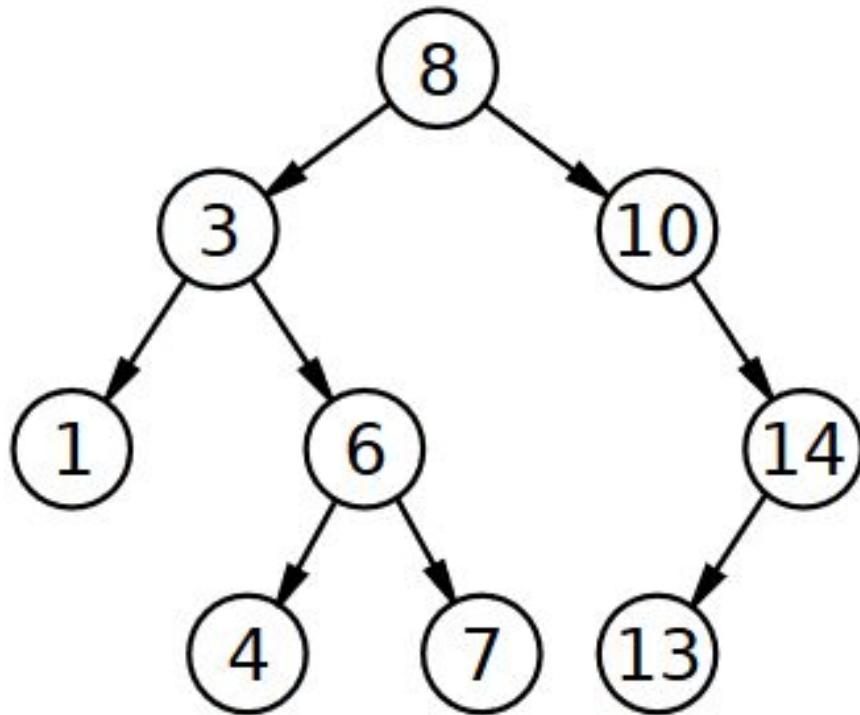
- sortuje elementy (klucze) wg porządku naturalnego (Comparable<E>) lub wg podanego komparatora

```
1  @Test
2  public void givenTreeMap_whenOrdersEntriesNaturally_thenCorrect() {
3      TreeMap<Integer, String> map = new TreeMap<>();
4      map.put(3, "val");
5      map.put(2, "val");
6      map.put(1, "val");
7      map.put(5, "val");
8      map.put(4, "val");
9
10     assertEquals("[1, 2, 3, 4, 5]", map.keySet().toString());
11 }
```

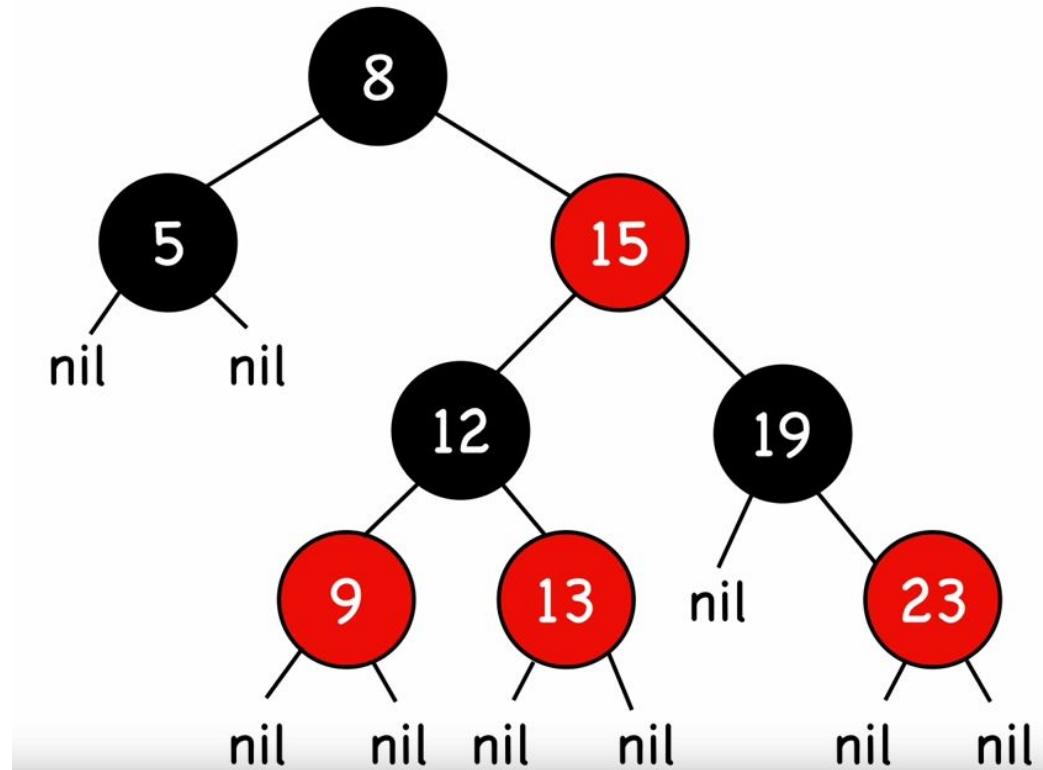
TreeMap (implements [Map<K,V>](#))

- sortuje elementy (klucze) wg porządku naturalnego (`Comparable<E>`) lub wg. podanego komparatora
- wykorzystuje drzewa binarne (czerwono-czarne)

Drzewo binarne



Red-black tree



Zadanie 10



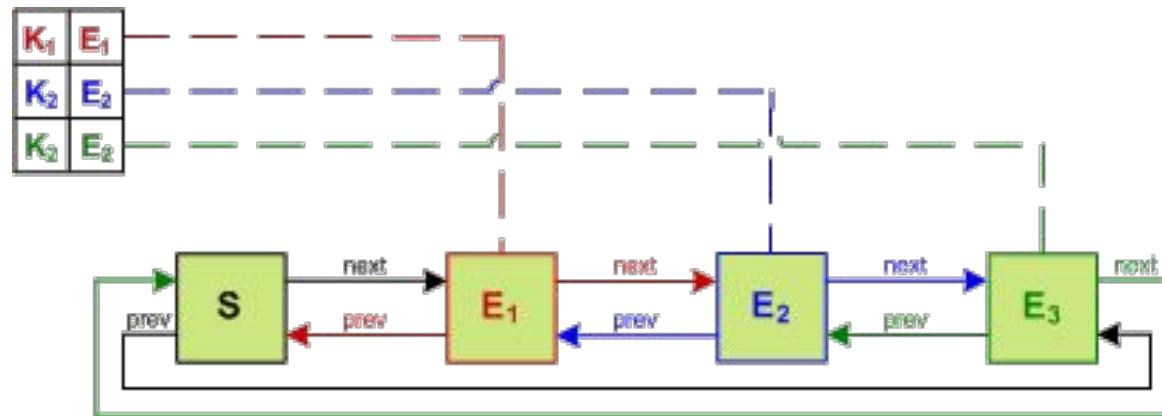
- 1.** W klasie `TreeMapTask` dodaj do `TreeMapy` kilka osób, załóż, że kluczem jest login. Dodaj przynajmniej dwie osoby o tym samym loginie.
- 2.** Pobierz jedną osobę używając jej loginu i wypisz jej dane.
- 3.** Iterując po kolekcji wypisz loginy oraz dane wszystkich osób. Porównaj z zadaniem 9 - jakie widzisz różnice?
- 4.** (*) Stwórz mapę (drzewo) podając własny komparator tak aby odwrócić kolejność sortowania drzewa.

HashMap vs TreeMap



W jakim przypadku powinniśmy użyć HashMap, a w jakim TreeMap?

LinkedHashMap



List/Set/Map

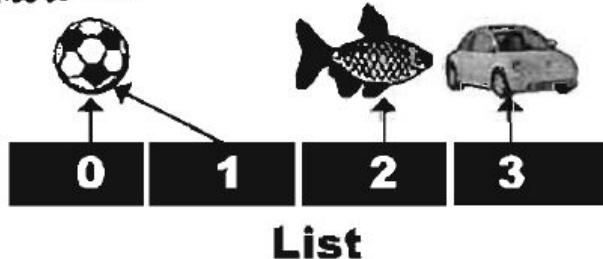
Powtóreczka

List/Set/Map - podsumowanie

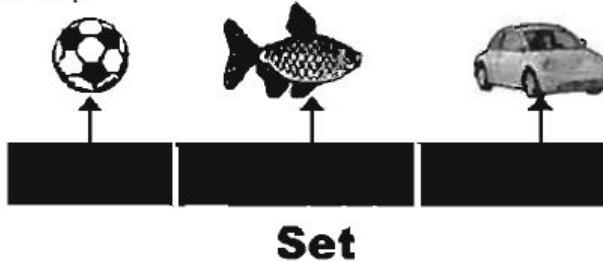
1. Czym się różni Set od Listy?

List/Set/Map - podsumowanie

Duplicates OK.



NO duplicates.



List/Set/Map - podsumowanie

1. Czym się różni Set od Listy?
2. Czym się różni Mapa od Listy?
3. Która z kolekcji nie zezwala na duplikaty?

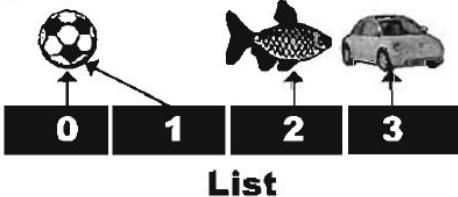
List/S

► LIST - when sequence matters

Collections that know about *Index position*.

Lists know where something is in the list. You can have more than one element referencing the same object.

Duplicates OK.

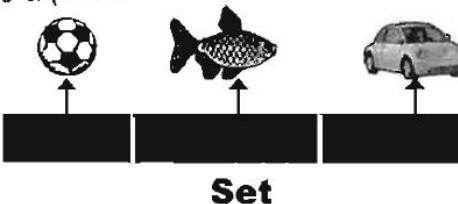


► SET - when uniqueness matters

Collections that *do not allow duplicates*.

Sets know whether something is already in the collection. You can never have more than one element referencing the same object (or more than one element referencing two objects that are considered equal—we'll look at what object equality means in a moment).

NO duplicates.

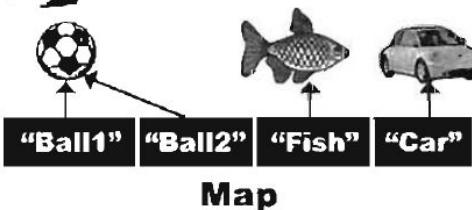


► MAP - when finding something by key matters

Collections that use *key-value pairs*.

Maps know the value associated with a given key. You can have two keys that reference the same value, but you cannot have duplicate keys. Although keys are typically String names (so that you can make name/value property lists, for example), a key can be any object.

Duplicate values OK, but NO duplicate keys.



List/Set/Map - podsumowanie

➤ **TreeSet**

Keeps the elements sorted and prevents duplicates.

➤ **HashMap**

Let's you store and access elements as name/value pairs.

➤ **LinkedList**

Designed to give better performance when you insert or delete elements from the middle of the collection. (In practice, an ArrayList is still usually what you want.)

List/Set/Map - podsumowanie

➤ **TreeSet**

Keeps the elements sorted and prevents duplicates.

➤ **HashMap**

Let's you store and access elements as name/value pairs.

➤ **LinkedList**

Designed to give better performance when you insert or delete elements from the middle of the collection. (In practice, an ArrayList is still usually what you want.)

➤ **HashSet**

Prevents duplicates in the collection, and given an element, can find that element in the collection quickly.

➤ **LinkedHashMap**

Like a regular HashMap, except it can remember the order in which elements (name/value pairs) were inserted, or it can be configured to remember the order in which elements were last accessed.

Kolejki (Queues)



PriorityQueue

A priority queue is an abstract data type (which is like a regular queue or stack data structure), but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time (depending on which constructor is used).

PriorityQueue

- przyjmuje elementy nieuporządkowane, a zwraca uporządkowane (o najmniejszej wartości)
- nie gwarantuje przechowywania elementów w kolejności

```
PriorityQueue<Integer> queue = new PriorityQueue<>();  
queue.add(8);  
queue.add(4);  
queue.add(10);  
  
Integer leastElement = queue.peek(); // 4
```

PriorityQueue

- peek - zwraca najmniejszy element (z ang. peek - zerkać)
- poll - zwraca i usuwa najmniejszy element
- add - dodaje element do kolejki
- i inne...

Zadanie 11



- w klasie QueueTask przetestuj działanie metod kolejki priorytetowej: add, peek i poll. Użyj porządku naturalnego
- (*) użyj komparatora zewn
- (*) przeiteruj po elementach kolejki iteratorem. Co zauważysz?

Czas



java.util.Date

- dla < Java 1.8
- przechowuje datę i czas
- metody:
 - ✓ setTime - ustawia czas (parametr to milisekundy od 1 stycznia 1970)
 - ✓ compareTo - porównywanie dat
 - ✓ before/after - sprawdza czy data jest wcześniejsza/późniejsza od parametru
 - ✓ i inne...

SimpleDateFormat

- klasa do formatowania `java.util.Date`
- konstruktor może przyjmować Pattern
- metody:
 - ✓ `format` - zwraca sformatowany tekst
 - ✓ `parse` - przekształca tekst na datę
 - ✓ i inne...

<https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html>

Calendar

- klasa reprezentująca kalendarz
- pozwala manipulować czasem

```
Calendar now = Calendar.getInstance();
now.add(Calendar.MONTH, amount: 1);
Date nextMonth = now.getTime();
```

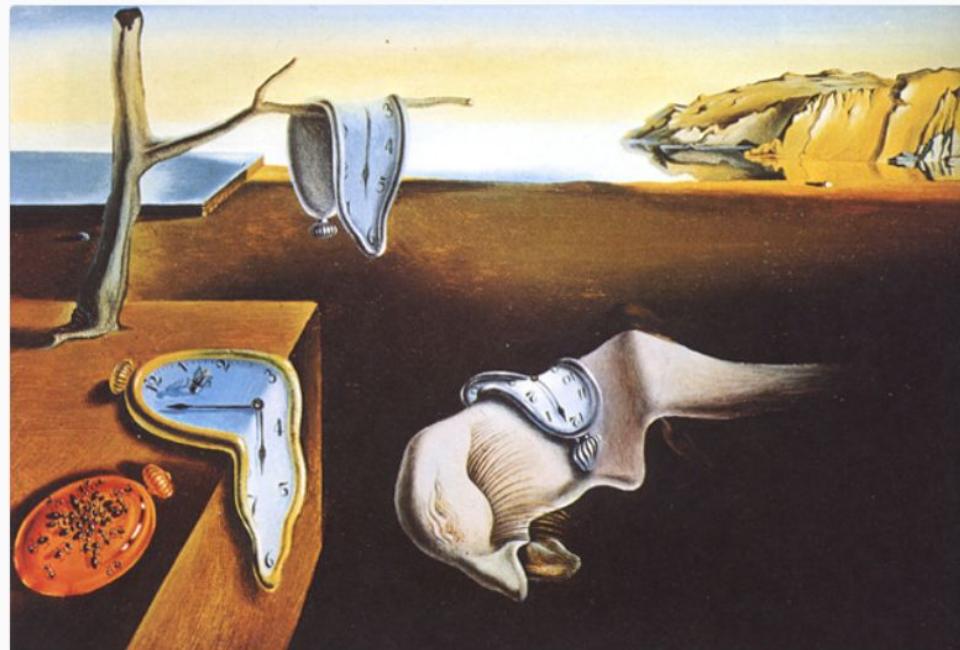
Zadanie 12



W klasie `DateTask` wypisz na konsolę sformatowany obiekt daty, który wskazuję na 2 tygodnie i 3 dni do przodu od dzisiaj. Ustaw godzinę 11:23.

Wybierz czytelny dla użytkownika format daty i wyświetl datę.

java.util.Date



"java.util.Date"

Salvador Dali

Oil on canvas, 1931

494 notes

... ↗ ❤

<http://classicprogrammerpaintings.com>

java.util.Date - wady

- problematyczna przy współbieżności
- mało praktyczna w codziennych operacjach
- sporo metod oznaczonych jako przestarzałe
- niełatwwe operowanie na strefach czasowych
- często zastępowane biblioteką Joda Time
<https://www.joda.org/joda-time/>

java.util.Date

```
Date date = new Date();
```

```
m  notify()                      void
m  notifyAll()                    void
m  wait()                        void
m  wait(long timeout)           void
m  wait(long timeout, int nanos) void
m  getDate()                     int
m  getDay()                      int
m  getHours()                    int
m  getMinutes()                  int
m  getMonth()                    int
m  getSeconds()                  int
```

Use Ctrl+Shift+Enter to syntactically correct your code after completing (balance parentheses etc.) >> 

```
date.
```

LocalDate/LocalDateTime

- dla Java 1.8+
- nowe, wygodniejsze API Javy
- więcej na zajęciach z Java 8+...

```
LocalDate now = LocalDate.now();  
LocalDate tommorow = now.plusDays(1);
```

Properties



Properties

- przechowuje pary klucz-wartość (tylko Stringi)
- unikalne klucze
- przydatna do przechowywania ustawień aplikacji
- można podać dodatkowe, domyślne Properties

Properties

- przechowuje pary klucz-wartość (tylko Stringi)
- unikalne klucze
- przydatna do przechowywania ustawień aplikacji
- można podać dodatkowe, domyślne Properties

```
Properties properties = new Properties();
properties.setProperty("key", "value");
String value = properties.getProperty("key");
```

Zadanie 13



- 1.** Stwórz plik config.properties w resources, dodaj do niego kilka par klucz-wartość rozzielone znakiem =
- 2.** W klasie PropertiesTask wczytaj plik (użyj metody load z obiektu Properties oraz klasy FileInputStream).
- 3.** Wypisz na konsolę wszystkie klucze z pliku.
- 4.** Wypisz na konsolę wartość dla jednego z kluczy.

Locale



Locale

- klasa obsługująca lokalizację geograficzną użytkownika
- często potrzebna do prawidłowego wyświetlania numerów (np. w Polsce z przecinkami, w USA z kropkami), dat
- dzięki niej wiadomo jaki dzień tygodnia jest pierwszy, jaki jest kod kraju, języka itp.

```
Locale constantLocale = Locale.JAPAN;  
Locale createdLocale = new Locale( language: "pl", country: "PL");
```

Zadanie 14



W klasie `DateTask` wypisz na konsolę datę sformatowaną zgodnie z kilkoma krajami.

Użyj `DateFormat.getDateInstance` oraz stałych formatu z klasy `DateFormat`.

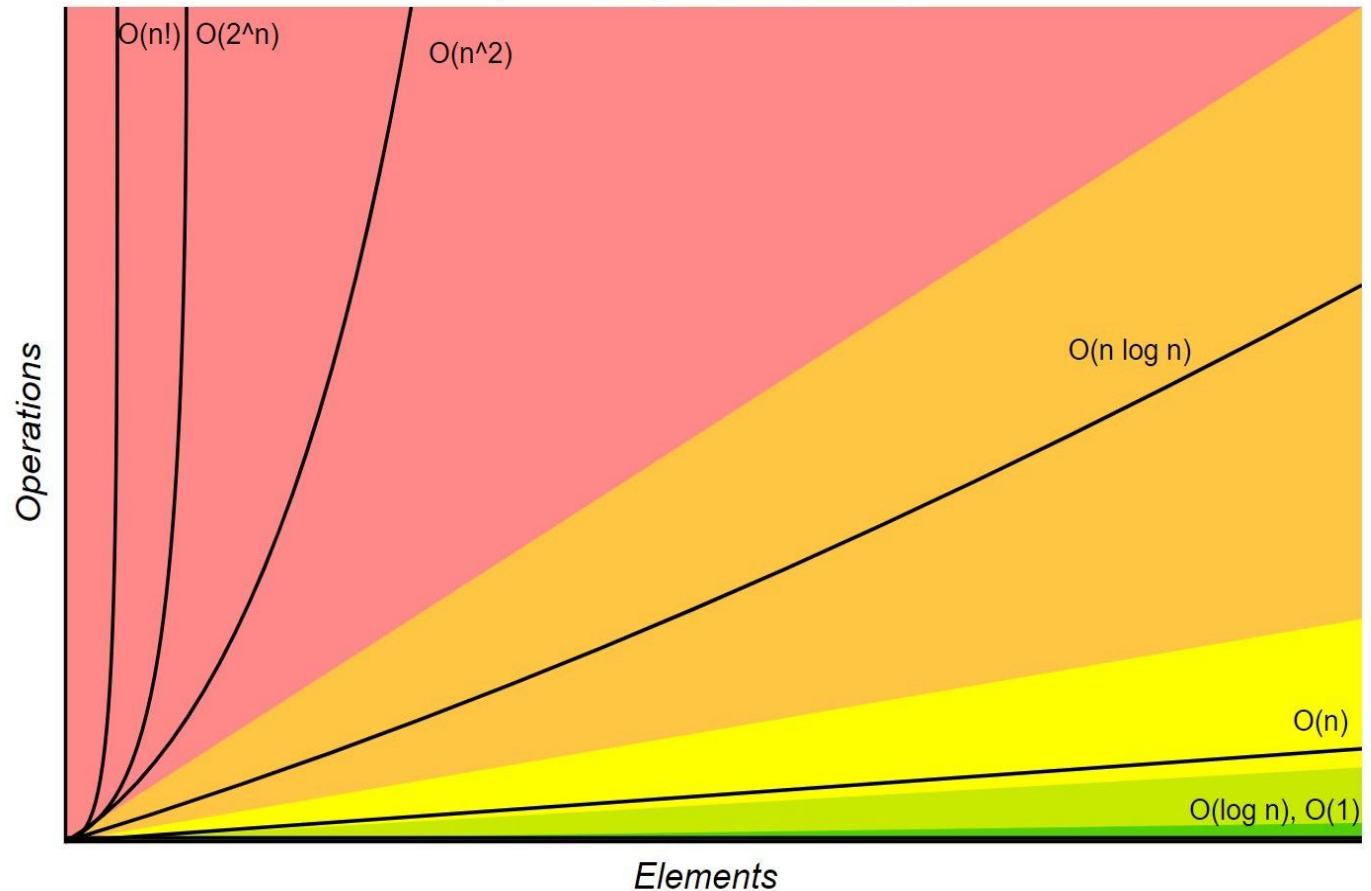
Złożoność obliczeniowa

This image shows a page filled with handwritten mathematical notes and calculations. The handwriting is in black ink, with some red highlights and circles used to emphasize certain parts of the text. The content covers a wide range of topics, including set theory, functions, and geometric concepts. There are several diagrams, such as a circle with points labeled S₁, S₂, and S₃, and a triangle with vertices labeled X₁, X₂, and X₃. The page is filled with complex formulas and equations, some of which are crossed out or have annotations. A large red circle is drawn around a specific area of the text, likely indicating a key concept or result.

Notacja Duże O

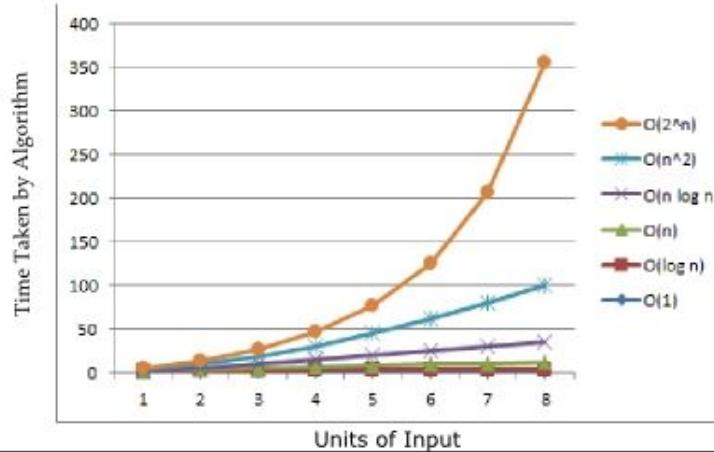
Big-O Complexity Chart

Horrible Bad Fair Good Excellent



Big O Notation

Big O notation is a measure of how long an algorithm takes to process an additional unit of input. The graph below is the time complexity of the most common cases, $O(1)$ through $O(2^n)$ over 1 through 8 input units.



Big-O notation	Computations for 10 elements	Computations for 100 elements	Computations for 1000 elements
$O(1)$	1	1	1
$O(\log n)$	3	7	10
$O(n)$	10	100	1000
$O(n \log n)$	33	664	9966
$O(n^2)$	100	10000	1000000
$O(2^n)$	1024	1.26765E+30	1.0715E+301
$O(n!)$	3628800	9.3326E+157	4.0238726E+2567

- złożoność stała
- złożoność logarytmiczna
- złożoność liniowa
- liniowo-logarytmiczna
- kwadratowa
- wykładnicza
- złożoność typu silnia

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

Dzięki

Pytania?

Materiały

https://www.youtube.com/watch?v=KyUTuwz_b7Q - Hash Tables and Hash Functions

<https://www.youtube.com/watch?v=oSWTXtMglKE> - Data Structures: Trees

<https://www.youtube.com/watch?v=qvZGUFWChY> - Red-black trees in 4 minutes – The basics

<http://www.informit.com/articles/article.aspx?p=368648&seqNum=2>

<https://www.samouczekprogramisty.pl/podstawy-zlozonosci-obliczeniowej/>

Literatura

[William J. Collins] - Data Structures and the Java Collections Framework

[Goodrich, Tamassia & Goldwasser] - Data Structures and Algorithms in Java

[Bert Bates and Kathy Sierra] - Head first java

<https://www.baeldung.com/java-treemap>

<https://www.coengodegebure.com/understanding-big-o-notation/>

https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781783988181/2/ch02lvl1sec18/big-o-notation