infoShare ACADEMY

Invest in Pomerania ACADEMY

INVEST IN POMERANIA ACADEMY

Fundusze Europejskie Program Regionalny

Rzeczpospolita Polska

URZĄD MARSZAŁKOWSKI WOJEWÓDZTWA POMORSKIEGO

Unia Europejska Europejski Fundusz Rozwoju Regionalnego

infoShare ACADEMY

Invest in Pomerania ACADEMY

Java: kolekcje i struktury danych

Fundusze Europejskie Program Regionalny

Rzeczpospolita Polska

URZĄD MARSZAŁKOWSKI WOJEWÓDZTWA POMORSKIEGO

Unia Europejska Europejski Fundusz Rozwoju Regionalnego

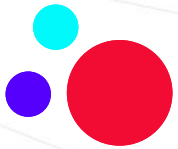
infoShareAcademy.com

HELLO

Tomasz Lisowski

Software developer
Scrum Master
IT trainer

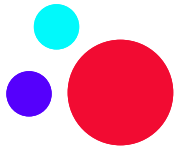
infoShare ACADEMY



Agenda

- wprowadzenie
- tablica
- lista
- set
- mapa

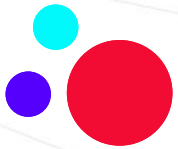
infoShare ACADEMY



Wprowadzenie

infoShareAcademy.com

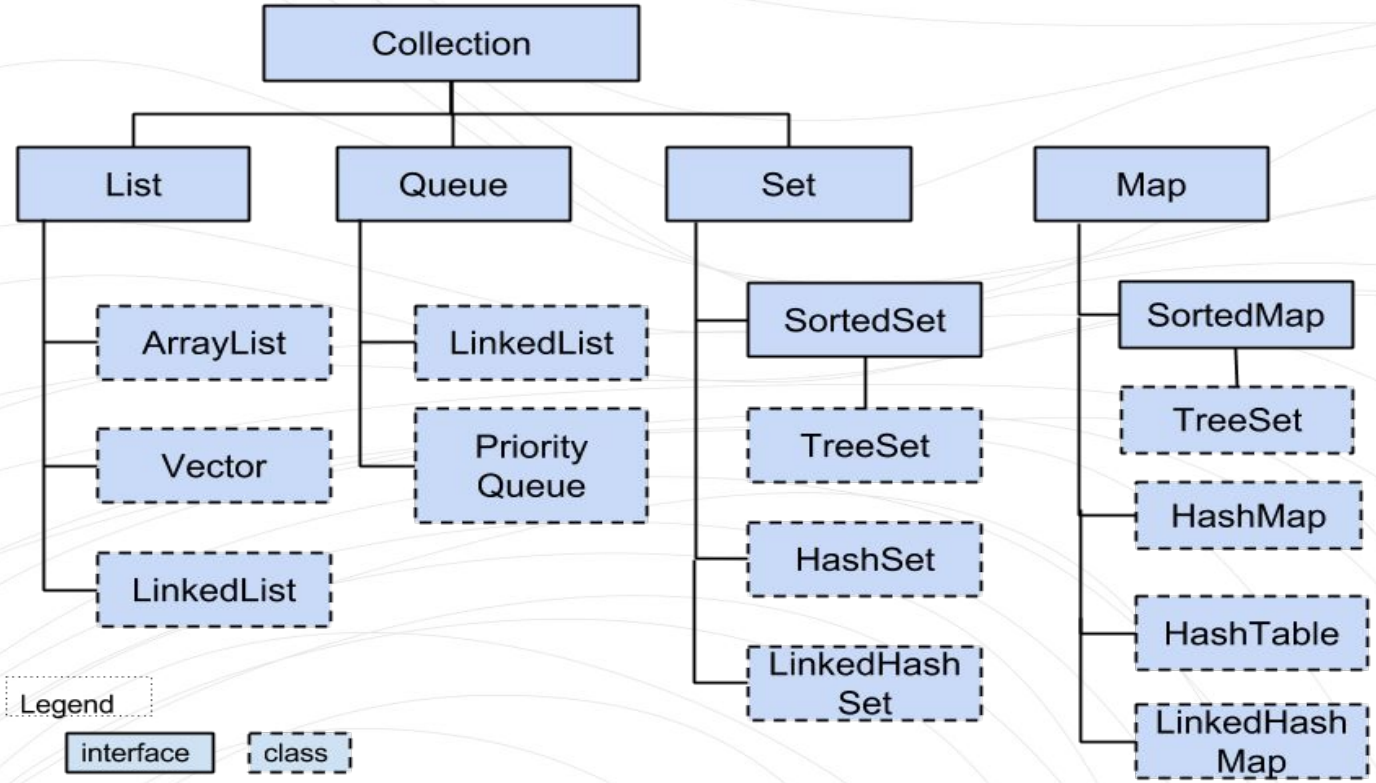
Co to są struktury danych?



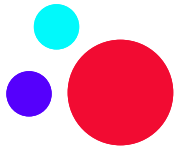
Struktury danych

- pojemniki na dane
- w sposób ustrukturyzowany przetrzymują informacje
- rekord - klasa
- *“sposób przechowywania danych w pamięci komputera. Na strukturach danych operują algorytmy”* (wikipedia)
- konkretna struktura na konkretny problem

Struktury danych



https://www.toolsqa.com/java/data-structure/



Tablica

Tablice

- struktura gromadząca uporządkowane dane
- tablice jedno lub wielowymiarowe
- wielkość jest stała (!)
- odwołanie do elementu po indeksie

typ[] nazwaTablicy = new typ[ilośćElementów]

typ[] nazwaTablicy = {el1, el2, ... , elN}

Tablice

- tworzenie tablicy
- odczyt danych
- przypisanie wartości do elementów tablicy

```
int[] tab = new int[3];
int[] tab2 = {1,2,3};

int number = tab[1];
int number2 = tab2[1];

tab[1] = 2;

tab[i] = 2;
```

Ćwiczenie

- stwórz metodę przyjmującą parametr typu int
- wewnątrz metody stwórz tablicę 10-elementową typu int
- uzupełnij tablicę kolejnymi liczbami całkowitymi, zaczynając od podanej w parametrze
- wypisz wszystkie elementy tablicy

np. parametr = 4

tablica: 4, 5, 6, ..., 13

Ćwiczenie 1a

- stwórz metodę przyjmującą parametr **Integer... params**
- wewnątrz metody:
  - stwórz nową tablicę typu **Integer** o rozmiarze **params.length**
  - iteruj po każdym elemencie zbioru *params*
  - dodaj każdy element do tablicy, ale pomnożony *2
  - zwróć wypełnioną tablicę
- przetestuj program

Ćwiczenie 1b

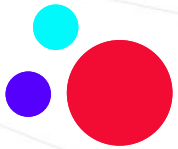
- stwórz metodę przyjmującą parametr **Car... cars**
- wewnątrz metody:
  - stwórz nową tablicę typu **Car** o rozmiarze **cars.length**
  - iteruj po każdym elemencie zbioru ***cars***
  - dodaj każdy element do tablicy
  - zwróć wypełnioną tablicę
- przetestuj program

Ćwiczenie 2

- stwórz metodę zwracającą tablicę obiektów typu Car
- skorzystaj z CarFactory i wygeneruj 5 samochodów
- metoda powinna “odfiltrować” wszystkie obiekty, których nazwa to “Fiat”
- zwróć tablicę pozostałych obiektów
- przetestuj program

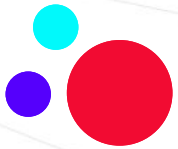

Lista


info Share
ACADEMY



Lista

- interfejs List
- każdy element ma przyporządkowany indeks
- możemy odwołać się do konkretnego elementu po indeksie
- obiekty mogą się powtarzać
- podstawowe operacje:
- add(object), get(index), remove(index), remove(object), size()



Lista

- **ArrayList** – przechowuje dane wewnątrz tablicy, wydajna gdy znamy ilość elementów lub wykonujemy mało operacji dodawania/usuwania
- **LinkedList** – przechowuje dane w postaci powiązanej, wydajniejsza gdy dodajemy/usuwamy dużo elementów

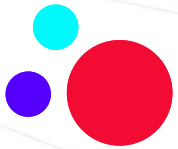
```
List<String> names = new ArrayList<>();
names.add("Andrzej");
names.add("Klaudia");
System.out.println(names.get(1)); //wypisze "Klaudia"
```

Ćwiczenie 3a

- stwórz kilka dowolnych obiektów (np. liczby lub słowa), w tym co najmniej dwa **równe**
- dodaj je do kolekcji typu **List**
- wypisz wszystkie elementy tej kolekcji

Ćwiczenie 3b

- stwórz kilka obiektów typu **Engine**, w tym co najmniej dwa **równe**
- dodaj je do kolekcji typu **List**
- wypisz wszystkie elementy tej kolekcji



Lista

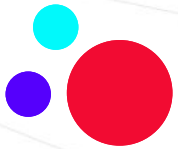
- obecność obiektu sprawdza się za pomocą metody **contains(..)**
- **indexOf(..)** zwraca nam indeks danego obiektu (-1 gdy brak)

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(3);

System.out.println(list.contains(1));    //true
System.out.println(list.contains(2));    //false
System.out.println(list.indexOf(1));     //0
System.out.println(list.indexOf(2));     //-1
```

Ćwiczenie 4

- stwórz listę obiektów typu **String**
- dodaj do niej obiekty: *“info”, “Share”, “Academy”*
- sprawdź, czy dana kolekcja zawiera napisy:
  - “info”
  - “INFO”
- wyświetl index obiektów:
  - “Academy”
  - “ISA”

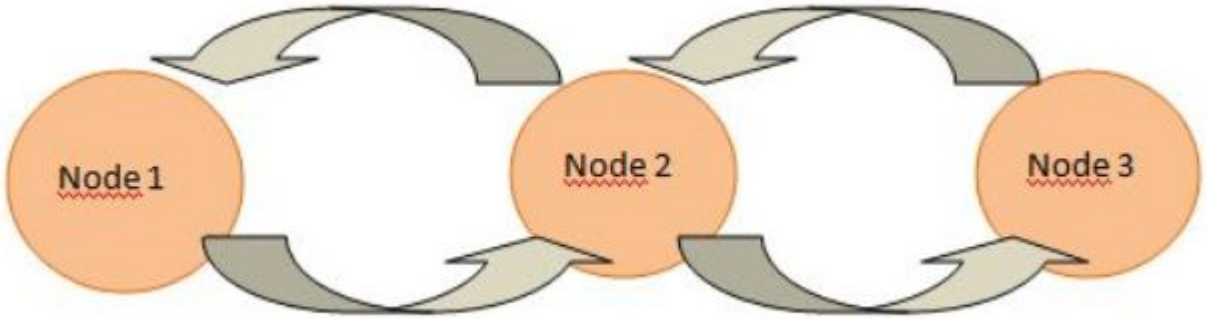


ArrayList

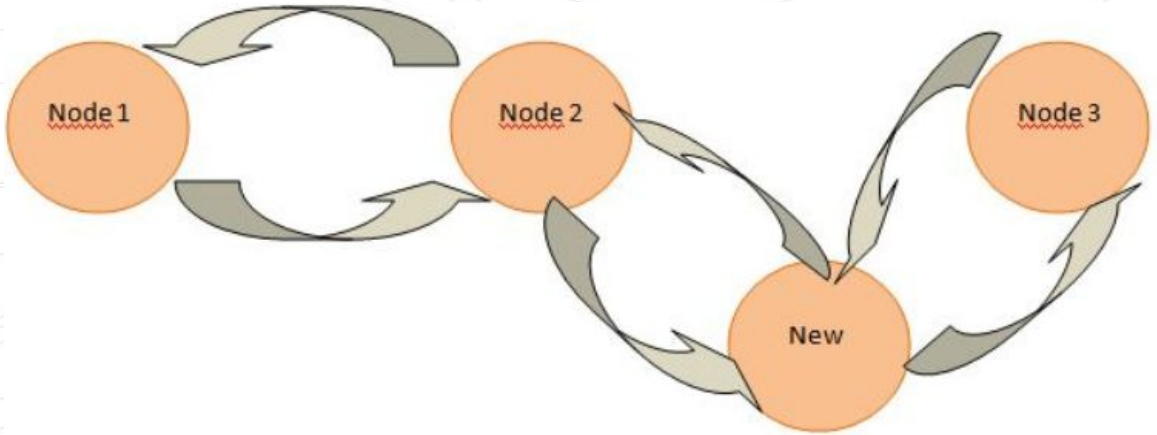
- może zawierać duplikaty
- zachowuje kolejność dodawania elementów
- posiada indeksy (“pod spodem” jest tablica)

LinkedList

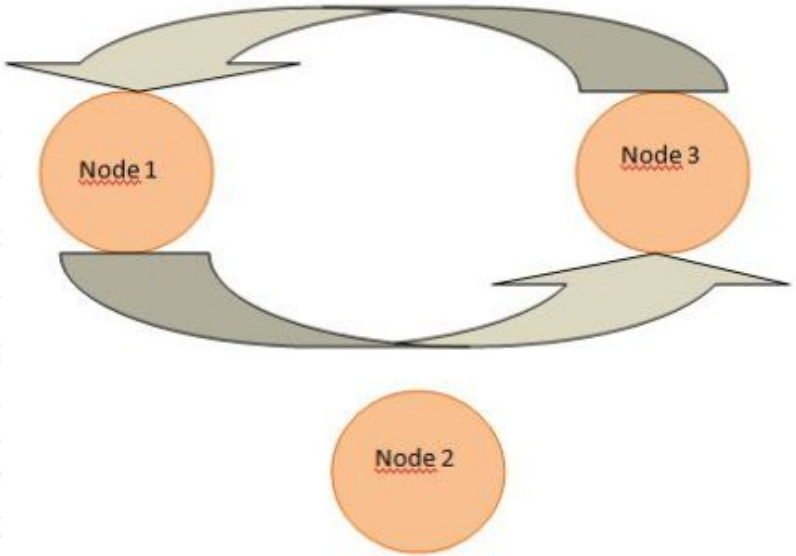
- składa się z węzłów (node), połączonych ze sobą
- każdy węzeł wie tylko o poprzednim i następnym węźle

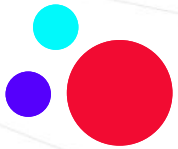


LinkedList - dodawanie elementu



LinkedList - usuwanie elementu





Lista - podsumowanie

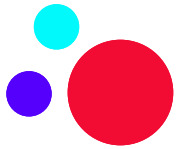
	ArrayList	**LinkedList**
get()	bardzo szybki	musi przejść całą listę
remove()	wolniejszy, ale nadal szybki	bardzo szybki
add()	wolniejszy, ale nadal szybki	bardzo szybki
kiedy używać?	prawie zawsze	w konkretnych przypadkach

Ćwiczenie 5

- stwórz metodę przyjmującą parametr ***Car... cars***
- wewnątrz metody:
  - stwórz kolekcję **List** typu **Car**
  - dodaj każdy element z ***cars*** do stworzonej kolekcji
  - zwróć wypełnioną listę
- prześlij do metody taki sam obiekt dwukrotnie
- wypisz wszystkie elementy otrzymanej kolekcji

Ćwiczenie 6

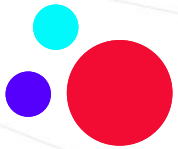
- stwórz metodę zwracającą **listę** obiektów typu **Car**
- skorzystaj z **CarFactory** i wygeneruj 10 samochodów
- metoda powinna “odfiltrować” wszystkie obiekty, których nazwa to “Fiat”
- zwróć listę pozostałych obiektów
- przetestuj program
- sprawdź rozmiar listy



Hashowanie

infoShareAcademy.com

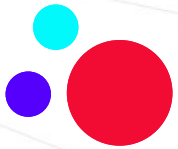
info Share ACADEMY



Hashowanie

- proces zamiany obiektu na “kod” typu **Integer**
- ten sam obiekt zawsze zwróci ten sam **hashCode**
- dwa równe obiekty zawsze mają ten sam **hashCode**
- .. ale dwa takie same hashCody nie zawsze oznaczają równość obiektów
- metody ***hashCode()*** i ***equals()*** są powiązane tzw. kontraktem:

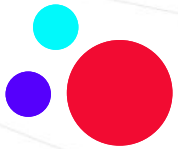
dla dwóch obiektów, których porównanie przy pomocy metody equals() zwraca true, metoda hashCode() powinna zwracać taką samą wartość



== *vs* equals

- instrukcje porównania
- **==** porównuje **referencję** (przestrzeń w pamięci)
- **equals()** porównuje **wartość** dwóch obiektów

- *domyślna implementacja *equals()* z klasy **Object**



== vs equals

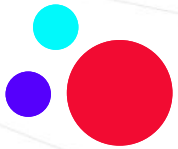
- equals() to metoda klasy Object

  jeśli obiekty są równe, to muszą mieć ten sam hashCode
  jeśli obiekty mają ten sam hashCode, to nie muszą być równe

- nadpisanie metody **hashCode()**
- kontrakt **hashCode()** $\Longleftrightarrow$ **equals()**

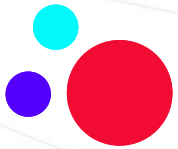

Set


info Share
ACADEMY



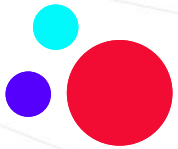
Set

- interfejs **Set**
- obiekty w zbiorze **nie mogą się powtarzać (!)**
- elementy nie mają przyporządkowanego indeksu (brak metody get())
- dostęp za pomocą iteratora
- nie gwarantuje kolejności elementów



Set

- **HashSet** – podstawowa implementacja, brak gwarancji kolejności, wymaga poprawnej implementacji hashCode() i equals()
- **TreeSet** – przechowuje elementy w postaci drzewa, uporządkowanie elementów zgodnie z Comparable/Comparator (sortowanie)
- **LinkedHashSet** - podobna do HashSet, ale gwarantuje kolejność elementów



Set

```
Set<String> names = new HashSet<>();
names.add("Andrzej");
names.add("Klaudia");
for (String name : names) {
    System.out.println(name);
}
```

Ćwiczenie 7a

- stwórz kilka dowolnych obiektów, w tym co najmniej dwa równe
- dodaj je do kolekcji **Set**
- wypisz wszystkie elementy tej kolekcji

Ćwiczenie 7b

- stwórz nową klasę (dwa pola typu **Integer**)
- stwórz kilka obiektów powyższej klasy, w tym co najmniej dwa równe
- dodaj je do kolekcji **Set**
- wypisz wszystkie elementy tej kolekcji
- czy kolekcja zawiera duplikaty?

Ćwiczenie 8

- stwórz set obiektów typu **String**
- dodaj do niej obiekty: *“info”*, *“Share”*, *“Academy”*
- sprawdź, czy dana kolekcja zawiera napisy:
  - “info”
  - “INFO”

Ćwiczenie 9

- stwórz metodę przyjmującą parametr ***Engine... engines***
- wewnątrz metody:
  - stwórz kolekcję **Set** typu **Engine**
  - dodaj każdy element z *engines* do stworzonej kolekcji
  - zwróć wypełniony set
- prześlij do metody taki sam obiekt dwukrotnie
- wypisz wszystkie elementy otrzymanej kolekcji


Mapa


infoShare
ACADEMY

Mapa

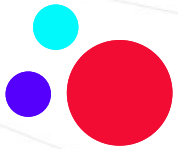
- formalnie nie są kolekcjami (nie są typu Collection)
- przechowują parę klucz-wartość
- do elementów odwołujemy się po kluczu
- .. który wskazuje na wartość
- klucz jest obiektem
- klucze muszą być unikalne
- podstawowe operacje:

  put(K, V), get(K), containsKey(K), keySet()

Mapa

- **HashMap** – właściwości podobne do HashSet
- **TreeMap** – elementy przechowywane w formie posortowanej (wg klucza)
- **LinkedHashMap** – zachowuje kolejność dodawania elementów

```
Map<String, Integer> map = new HashMap<>();
map.put("pierwszy", 1);
map.put("drugi", 2);
System.out.println(map.get("pierwszy")); //wypisze liczbę 1
Set<String> keys = map.keySet();
Collection<Integer> values = map.values();
```



Ćwiczenie 10a

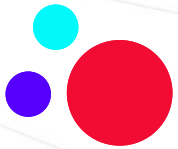
- stwórz kilka obiektów (**Integer**), w tym dwa równe (**klucze**)
- stwórz kilka obiektów (**String**), w tym dwa równe (**wartości**)
- dodaj elementy do mapy
- wypisz wszystkie elementy tej kolekcji:
  - klucze
  - wartości
  - pary: klucz - wartość

Ćwiczenie 10b

- stwórz obiekty typu Integer - 1, 2, 3, 5
- stwórz obiekty typu String - *“ISA”, “info”, “Share”, “Java”*
- dodaj obiekty do mapy ***Map<Integer, String>***
- wyświetl wartości spod kluczy 4 i 5
- sprawdź czy w mapie istnieje klucz 15
- sprawdź czy w mapie istnieje wartość “ISA”

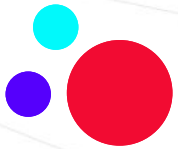
Ćwiczenie 11a

- stwórz metodę przyjmującą parametr ***Engine... engines***
- wewnątrz metody:
  - stwórz kolekcję **Map** typu **<Integer, Engine>**
  - kluczem jest moc silnika
  - dodaj każdy element z engines do stworzonej kolekcji (pod odpowiednim kluczem)
  - zwróć wypełnioną mapę
- prześlij do metody taki sam obiekt dwukrotnie
- wypisz wszystkie elementy otrzymanej kolekcji



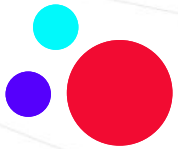
Ćwiczenie 11b

- stwórz metodę przyjmującą parametr ***Car... cars***
- wewnątrz metody:
  - stwórz kolekcję **Map** typu **<String, Car>**
  - kluczem jest nazwa pojazdu
  - dodaj każdy element z cars do stworzonej kolekcji (pod odpowiednim kluczem)
  - zwróć wypełnioną mapę
- prześlij do metody taki sam obiekt dwukrotnie
- wypisz wszystkie elementy otrzymanej kolekcji



Struktury danych

tablica	set	lista	mapa
uporządkowane dane	dane ułożone “losowo”	dane uporządkowane	pary klucz-wartość
stała wielkość	elementy nie mogą się powtarzać	elementy mogą się powtarzać	klucze muszą być unikalne (put() nadpisze wartość)
każdy el. ma swój indeks	brak indeksów	każdy el. ma swój indeks	klucze to set; wartość to dowolny obiekt, dostęp po kluczu

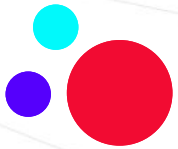


Mapa

- klucz i wartość to typy obiektowe
- zarówno jednym i drugim może być dowolny obiekt
- kolekcje to też obiekty
- możliwa jest mapa, w której kluczem jest inna mapa
- dodanie istniejącego klucza nadpisze jego wartość

Ćwiczenie 12

- stwórz metodę przyjmującą parametr ***Car... cars***
- wewnątrz metody:
  - stwórz kolekcję **Map** typu **<Integer, List<Car>>**
  - kluczem jest pojemność silnika danych pojazdów
  - dodaj każdy element z cars do stworzonej kolekcji (pod odpowiednim kluczem)
  - zwróć wypełnioną mapę
- prześlij do metody taki sam obiekt dwukrotnie
- wypisz wszystkie elementy otrzymanej kolekcji



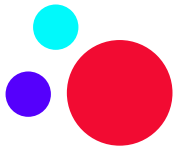
Struktury danych - materiały dodatkowe

- https://stormit.pl/struktury-danych/
- https://www.kodolamacz.pl/blog/wyzwanie-java-4-algorytmy-i-struktury-danych-w-jezyku-java/

infoShare ACADEMY

Q&A

infoShareAcademy.com



Thanks!


tomasz.lisowski@proton.me


www.infoshareacademy.com

infoShare ACADEMY

infoShare ACADEMY

Invest in Pomerania ACADEMY

INVEST IN POMERANIA ACADEMY

Fundusze Europejskie Program Regionalny

Rzeczpospolita Polska

URZĄD MARSZAŁKOWSKI WOJEWÓDZTWA POMORSKIEGO

Unia Europejska Europejski Fundusz Rozwoju Regionalnego