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

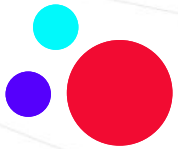
infoShareAcademy.com

HELLO

Tomasz Lisowski

Software developer
Scrum Master
IT trainer

infoShare ACADEMY

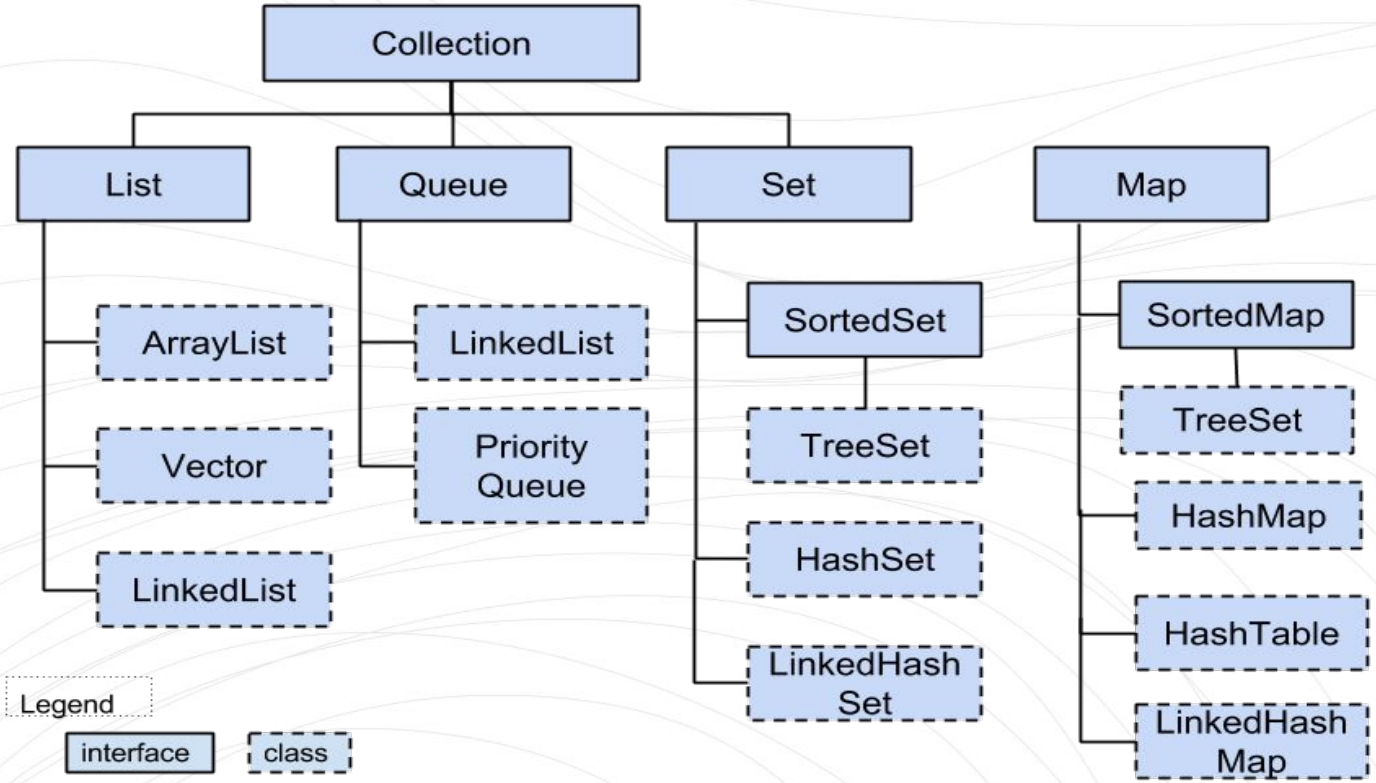


Agenda

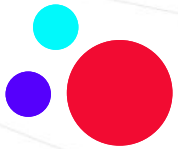
- powtórka
- mapa
- Iterator
- Comparator / Comparable
- Properties
- JSON

infoShare ACADEMY

Struktury danych

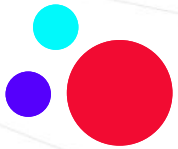


https://www.toolsqa.com/java/data-structure/



Lista

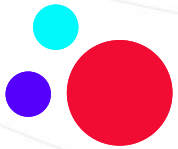
- interfejs List
- każdy element ma przyporządkowany indeks
- możemy odwołać się do konkretnego elementu po indeksie
- obiekty mogą się powtarzać
- podstawowe operacje:
- add(object), get(index), remove(index), remove(object), size()



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"
```



Lista

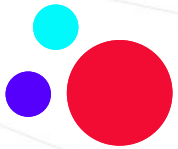
- obecność obiektu sprawdza się za pomocą metody **contains(..)**
- **indexOf(..)** zwraca nam indeks danego obiektu (-1 gdy brak)

```
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 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

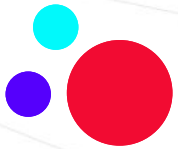


== vs equals

- equals() to metoda klasy Object

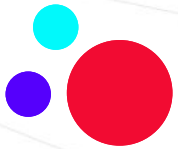
  jeśli obiekty są równe, to muszą mieć ten sam hashCode
  jeśli obiekty mają ten sam hashCode, to nie muszą być równe

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



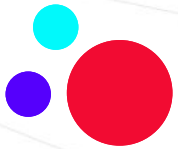
Set

- 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 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

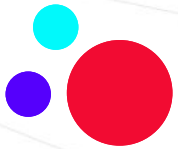
- formalnie nie są kolekcjami (nie są typu Collection)
- przechowują parę klucz-wartość
- do elementów odwołujemy się po kluczu
- .. który wskazuje na wartość
- klucz jest obiektem
- klucze muszą być unikalne
- podstawowe operacje:

  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

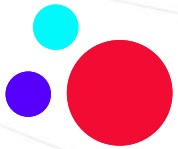
- stwórz kilka obiektów (**Integer**), w tym dwa równe (**klucze**)
- stwórz kilka obiektów (**String**), w tym dwa równe (**wartości**)
- dodaj elementy do mapy
- wypisz wszystkie elementy tej kolekcji:
  - klucze
  - wartości
  - pary: klucz - wartość

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”

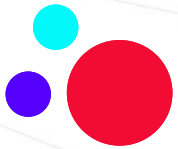
Ćwiczenie 11a

- 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



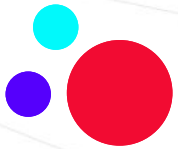
Ćwiczenie 11b

- 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

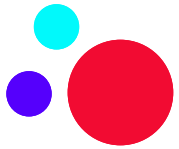


Mapa

- 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



Iterator / Comparator

infoShareAcademy.com

infoShare ACADEMY

Iterator

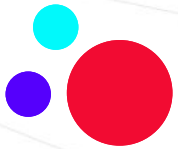
- obiekt tworzony na podstawie kolekcji
- pozwala na przejście po całej kolekcji
- i modyfikację jej elementów (nawet usuwanie)
- każda klasa z pakietu **Collection** jest “Iterable”

```
public interface Collection<E> extends Iterable<E> {
```

- główne metody:
  hasNext(), next(), remove()

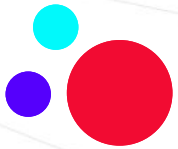
Ćwiczenie 13

- stwórz kilka obiektów Car i dodaj je do listy
- stwórz iterator
- użyj pętli while i metody hasNext()
- wypisz wszystkie elementy za pomocą iteratora
- usuń ostatni element z listy za pomocą iteratora



Comparable

- interfejs służący do sortowania danych konkretnego typu
- wymaga implementacji metody ***compareTo()***
- metoda ta odpowiada za logikę porównania obiektów
- wynik porównania:
  - **-1 (ujemna)** – obiekt pierwszy (na którym wołamy metodę) jest mniejszy
  - **1 (dodatnia)** – obiekt pierwszy (na którym wołamy metodę) jest większy
  - **0** – obiekty są równe
- sortowanie kolekcji za pomocą metody: **Collections.sort(collection);**



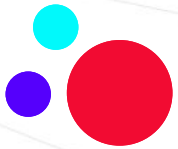
Comparable

```
public class Car implements Comparable {


    @Override
    public int compareTo(Object o) {
```

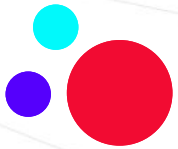
Ćwiczenie 14

- niech klasa Engine implementuje interfejs **Comparable**
- “większy” silnik to ten, który ma większą moc (pole power)
- stwórz kilka obiektów typu Engine i dodaj je do listy
- wypisz elementy z listy
- posortuj kolekcję
- wypisz elementy ponownie



Comparator

- klasa definiująca konkretny sposób sortowania
- musi implementować interfejs **Comparator**
- posiada metodę **compare(Type o1, Type o2)**
- wynik porównania:
  - **-1 (ujemna)** – obiekt pierwszy (na którym wołamy metodę) jest mniejszy
  - **1 (dodatnia)** – obiekt pierwszy (na którym wołamy metodę) jest większy
  - **0** – obiekty są równe
- sortowanie za pomocą metody:
  Collections.sort(collection, comparator);



Comparator

```
public class PowerComparator implements Comparator<Car> {


    @Override
    public int compare(Car o1, Car o2) {
```

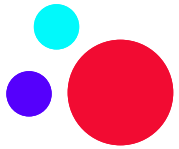
Ćwiczenie 15a

- zobacz jak wygląda klasa **PowerComparator** porównująca obiekty typu **Car** wg pola **power** ich silnika
- przygotuj listę kilku obiektów typu **Car**
- wypisz je na ekran jeden pod drugim
- posortuj dane za pomocą **PowerComparator**
- wypisz dane ponownie
- porównaj wyniki

Ćwiczenie 15b

- stwórz własny **comparator** porównujący klasę **Car** wg **capacity**
- przygotuj listę kilku obiektów typu **Car**
- wypisz je na ekran jeden pod drugim
- posortuj dane za pomocą własnego comparatora
- wypisz dane ponownie
- porównaj wyniki

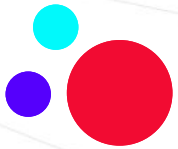
- *stwórz kolejny **comparator** - wg pola **maxSpeed**
- *posortuj dane za pomocę nowego comparatora



Properties

infoShareAcademy.com

infoShare ACADEMY



Properties

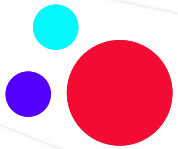
- przechowuje pary klucz-wartość (Stringi)
- unikalne klucze
- przydatna do przechowywania ustawień aplikacji
- można tworzyć własne obiekty property
- ***load()*** - metoda wczytująca dane (InputStream)

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

Ćwiczenie 16

- stwórz katalog resources (jeśli nie masz go w projekcie)
- stwórz w nim plik **config.properties**
- dodaj do pliku wartości w formacie klucz=wartosc
- wczytaj plik konfiguracyjny (***new InputStream(pathString)***)
- wyświetl wszystkie klucze
- wyświetl wartość dla jednego, dowolnego klucza
- wyświetl wszystkie dane (klucze i wartości)

infoShare ACADEMY



JSON

- lekki format wymiany danych
- składnia JS (JavaScript Object Notation)
- łatwy do zrozumienia dla człowieka
- łatwy do parsowania/generowania przez maszynę
- służy do komunikacji pomiędzy różnymi modułami lub systemami
- może przechowywać proste dane jak i bardzo złożone

JSON

- wspierane typy danych to:

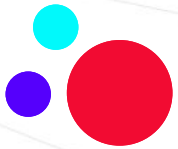
  String, boolean, number, arrays, Object
- obiekt JSON jest ‘zamknięty’ w klamry { }, tak jak klasa Java

```
{
  "name":"John",
  "age":30,
  "car":null
}
```

infoShare ACADEMY

JSON

- **serializacja** - zamiana obiektu Java na obiekt JSON
- **deserializacja** - zamiana obiektu JSON na obiekt Java



Jackson

- *com.fasterxml.jackson*
- popularna biblioteka do pracy z formatem JSON
- **ObjectMapper** - obiekt służący do mapowania

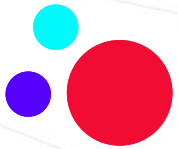
  - **writeValue(file, object)** - zapis obiektu do pliku
  - **writeValueAsString(object)** - serializacja obiektu do String
  - **readValue(file, Object.class)** - odczyt obiektu z pliku
  - **readValue(jsonString, Object.class)** - deserializacja na obiekt
  - **writeWithDefaultPrettyPrinter()** - formatuje wynik

Ćwiczenie 17a

- stwórz obiekt Engine z przykładowymi danymi
- korzystając z **ObjectMapper** zamień powyższy obiekt na String
- wypisz sformatowany wynik
- stwórz nowy obiekt Engine na podstawie powyższego Stringa
- wypisz i porównaj obydwa obiekty Engine

Ćwiczenie 17b

- stwórz kilka obiektów Engine z przykładowymi danymi
- umieść powyższe obiekty w tablicy
- korzystając z **ObjectMapper** zamień tablicę na String
- wypisz sformatowany wynik
- stwórz tablicę obiektów Engine na podstawie powyższego Stringa
- wypisz i porównaj wyniki



Gson

- *com.google.code.gson*
- **GsonBuilder** - builder obiektów typu Gson
- **Gson** - obiekt służący do mapowania

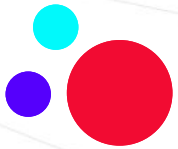
  - **toJson(object)** - serializacja obiektu do Stringa
  - **fromJson(jsonString, Object.class)** - deserializacja na obiekt

Ćwiczenie 18a

- stwórz obiekt Engine z przykładowymi danymi
- korzystając z **Gson** zamień powyższy obiekt na String
- wypisz sformatowany wynik
- stwórz nowy obiekt Engine na podstawie powyższego Stringa
- wypisz i porównaj obydwa obiekt Engine

Ćwiczenie 18b

- stwórz kilka obiektów Engine z przykładowymi danymi
- umieść powyższe obiekty w tablicy
- korzystając z **Gson** zamień tablicę na String
- wypisz sformatowany wynik
- stwórz tablicę obiektów Engine na podstawie powyższego Stringa
- wypisz i porównaj wyniki



Jackson *vs* Gson

- projekty open-source
- dla prostych obiektów Gson jest “prostszy” w obsłudze
- Jackson jest wbudowany w JAX-RS oraz Spring
- najpopularniejsze biblioteki do serializacji / deserializacji JSON
- https://www.baeldung.com/jackson-vs-gson

Ćwiczenie 19a

- stwórz obiekt Engine
- zamień go na obiekt JSON (dowolny sposób)
- zapisz sformatowany JSON do pliku na dysku
- odczytaj ten plik i odtwórz obiekt na jego podstawie
- wypisz dane tak utworzonego obiektu

- *zmodyfikuj zawartość zapisanego pliku w notatniku
- *uruchom program ponownie (tylko z opcją odczytu)

Ćwiczenie 19b*

- stwórz tablicę obiektów Engine
- zamień ją na obiekt JSON (dowolny sposób)
- zapisz sformatowany JSON do pliku na dysku
- odczytaj ten plik i odtwórz tablicę na jego podstawie
- wypisz wszystkie elementy z odczytanej tablicy

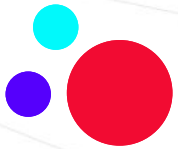
- *zmodyfikuj zawartość zapisanego pliku w notatniku
- *uruchom program ponownie (tylko z opcją odczytu)

Ćwiczenie 20a

- stwórz obiekt typu Car
- w dowolny sposób zamień go na String w formacie JSON
- wypisz sformatowany wynik
- odtwórz obiekt Car na podstawie powyższego Stringa
- wypisz i porównaj wyniki
- czy obydwa obiekty są sobie równe?

Ćwiczenie 20b

- stwórz listę obiektów typu Car
- w dowolny sposób zamień listę na String w formacie JSON
- wypisz sformatowany wynik
- odtwórz listę obiektów Car na podstawie powyższego Stringa
- wypisz i porównaj wyniki
- czy obydwie kolekcje są sobie równe?

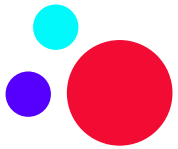


Struktury danych - materiały dodatkowe

- https://stormit.pl/struktury-danych/
- https://www.kodolamacz.pl/blog/wyzwanie-java-4-algorytmy-i-struktury-danych-w-jezyku-java/
- https://www.geeksforgeeks.org/comparable-vs-comparator-in-java

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