Kolekcje

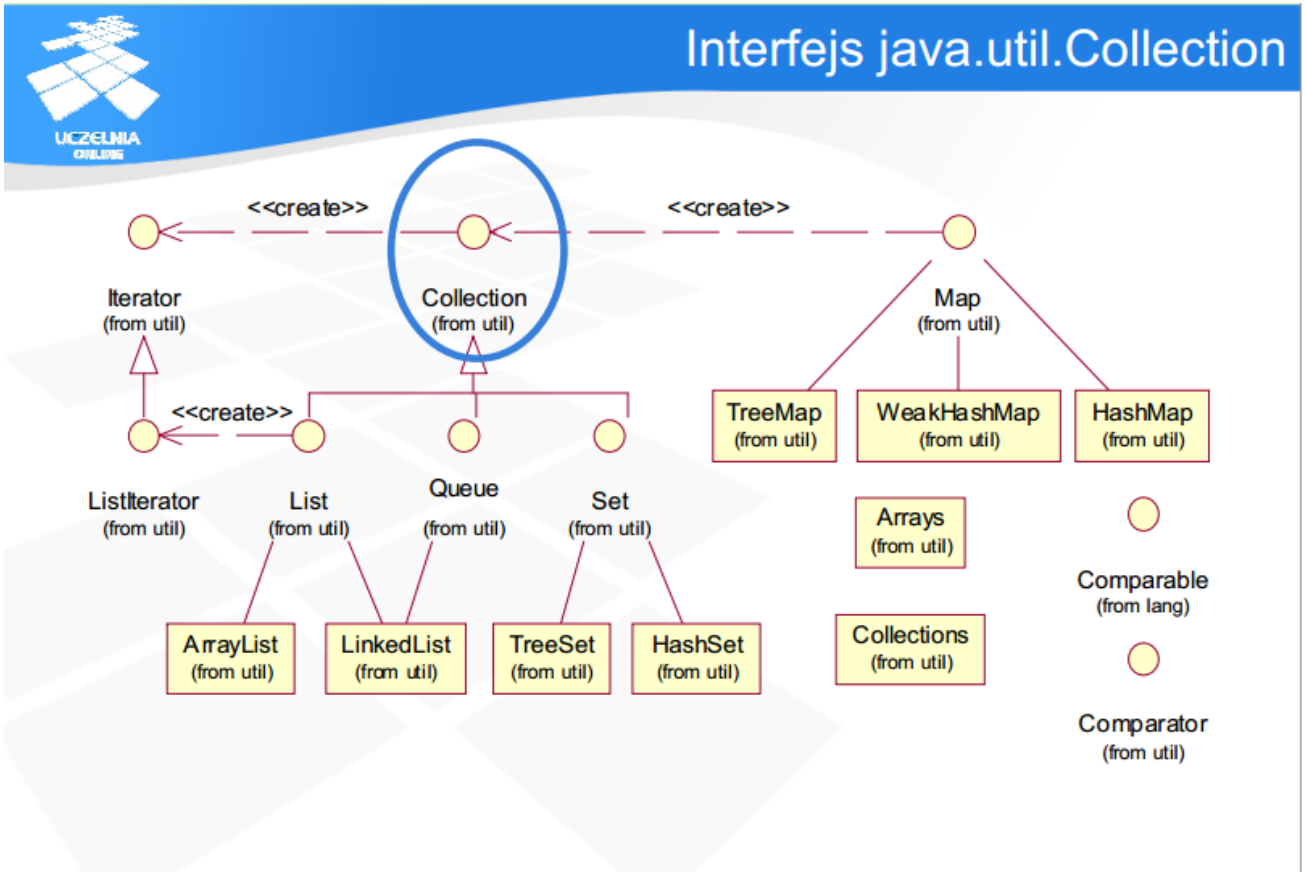
Kolekcje w Javie

- Kolekcja (kontener) – to po prostu obiekt, który grupuje wiele elementów w jeden twór.
- Pozwala na zapis, odczyt, korzystanie z danych oraz ich wzajemną komunikację.
- Przykład: ręka pokerowa (kolekcja kart), skrzynka odbiorcza (kolekcja maili), książka telefoniczna (słownik nazwisko-numer)

Po co?

Mniejszy koszt	• Jak jest potrzebna lista/stos/kolejka, to po co ją programować od zera? Lepiej skoncentrować się na ważnych elementach!
Zwiększa jakość i szybkość działania	• Stworzona kolekcja jest zoptymalizowana. Więcej: istnieją różne implementacje tej samej kolekcji (np. dla listy: ArrayList' oraz LinkedLista) zoptymalizowane pod konkretne zastosowania.
Łatwiejsze wzajemne oddziaływanie	• Kolekcje mają zdefiniowane standardowe podejścia.
Łatwiej się nauczyć jednego API	• Większość z modyfikatorów jest zgodna pomiędzy różnymi rodzajami kolekcji. Przejście z jednego rodzaju na drugi jest bezbolesne.
Łatwiej zaprojektować API	• Po co wynajdować koło od nowa skoro można użyć standardowych rozwiązań?
Łatwiej o re-use kodu	• Kolekcje z natury są łatwe do ponownego wykorzystania

java.util.Collection



Operacje wspólne

Operacje podstawowe – wykonywane na obiektach

- boolean add(Object obj)
- boolean remove(Object obj)
- boolean contains(Object obj)

Operacje grupowe – wykonywane na kolekcjach obiektów

- boolean addAll(Collection coll)
- boolean removeAll(Collection coll)
- boolean containsAll(Collection coll)
- boolean retainAll(Collection coll)
- void clear()

Operacje wspólne

Inspekcja

- int size()
- boolean isEmpty(Object obj)

Iteracja

- Iterator iterator()

Operacje na tablicach – konwersja do tablic

- Object[] toArray()
- Object[] toArray(Object[] type)

Lista

- Duplikaty możliwe
- Dostęp sekwencyjny
- Przykłady: ArrayList, LinkedList, Vector,

Ala	ma	kota	i	czipsy
0	1	2	3	4

Lista

Ala	ma	kota	i	czipsy
0	1	2	3	4

Dostęp pozycyjny do elementów

- Object get(int indeks)
- Object set(int indeks)
- Object add(int indeks)
- Object remove(int indeks)

Wyszukiwanie

- int indexOf(Object obiekt)
- int lastIndexOf(Object obiekt)

Widok przedziałowy

- List subList(int poczatek, int koniec)

Queue

Taka zwykła kolejka (LIFO, FIFO, priorytetowa) ☺

Dodawanie elementu	• boolean offer(Object obiekt)
Usuwanie elementu	• Object remove() • Object poll()
Inspekcja	• Object element() • Object peek()

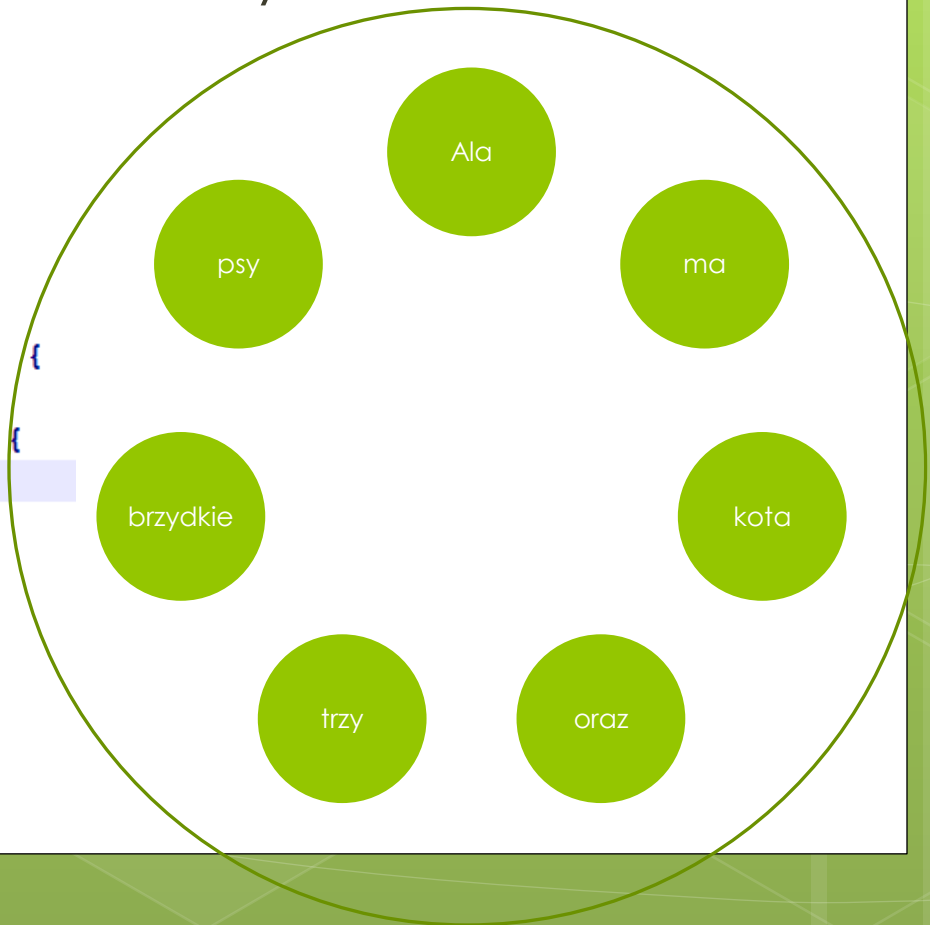
Queue - przykład

```
1  public class Oceny {
2      public static void main(String[] args) {
3      int liczba = Integer.parseInt(args[0]);
4      Queue kolejka = new LinkedList();
5          for (int i = liczba-1; i > 0; i--) {
6          kolejka.add(args[i]);
7          }
8      while (!kolejka.isEmpty()) {
9          System.out.println(kolejka.remove());
10         }
11     }
12 }
```

Set

- Reprezentacja zbioru matematycznego
- Brak uporządkowania elementów
- Dostęp sekwencyjny niezalecany
- Brak duplikatów

```
1  public class Duplikaty {
2      public static void main(String args[]) {
3      Set s = new HashSet();
4      for (int i = 0; i < args.length; i++) {
5          if (! s.add(args[i]))
6          System.out.println("Duplikat!");
7          }
8      }
9  }
```



Map

- Jednoznaczne odwzorowanie jak w słowniku
- Klucze, wartości i pary klucz-wartość d iako obiekty Collection

Map

Operacje podstawowe

- Object put(Object klucz, Object wartosc)
- Object get(Object klucz)
- Object remove(Object klucz)
- boolean containsKey(Object klucz)
- boolean containsValue(Object wartosc)

Operacje grupowe

- void putAll(Map zrodlo)

widoki-kolekcje

- Set keySet()
- Collection values()
- Set entrySet()

Map - przykłady

```
1  public class Czestotliwosc {
2  private static final Integer JEDEN = new Integer(1);
3      public static void main(String args[]) {
4      Map mapa = new HashMap();
5          for (int i=0; i<args.length; i++) {
6          Integer czest = (Integer) mapa.get(args[i]);
7          mapa.put(args[i], (czest == null ? JEDEN :
8          new Integer(czest.intValue() + 1)));
9  }
10     System.out.println(mapa.size() + " różnych słów");
11     System.out.println(mapa);
12 }
13 }
```

Iteratory

- Swoiste „wskaźniki”
- Służą do sekwencyjnego przeglądania wszystkich obiektów w kolekcji

```
1  public class Iteracja {
2  public static void main(String args[]) {
3  Collection lista = new ArrayList();
4  for (int i = 0; i < args.length; i++) {
5      lista.add(args[i]);
6  }
7  for (Iterator it = lista.iterator(); it.hasNext();)
8  String element = (String) it.next();
9  System.out.println("element = " + element);
10 }
11 }
12 }
13
14 > java Iteracja Ola i Ala to kolezanki
15 element = Ola
16 element = i
17 element = Ala
18 element = to
19 element = kolezanki
```

Algorytmy

- Metody statyczne klasy Collections implementują typowe algorytmy wykonywane na kolekcjach:
  - wyszukiwanie binarne
  - sortowanie
  - operacje algebraiczne na zbiorach
  - odwracanie list
  - permutacje list
  - wyszukiwanie elementów max/min

Sortowanie - porównanie

- Aby móc sortować, trzeba wiedzieć który obiekt jest „wcześniej”, a który „później”. Matematyk by powiedział o relacji częściowego porządku.
- Intuicyjnie wiemy, że 3 > 2
- Ale kto jest „większy” – Ala czy Ola?
- Java Collections korzysta z dwóch interfejsów służących do porównywania obiektów
  - java.lang.Comparable
  - java.util.Comparator

Interfejs Comparable

```
2  public class Person implements Comparable {
3   private Date birthday;
4  public Date getBirthday() {
5       return birthday;
6       }
7
8  public int compareTo (Object obj) {
9       String birthday = ((Person) obj).getBirthday();
10          return (this.birthday.compareTo(birthday));
11 }
12
13 Wywołanie: Collections.sort(people)
```

compareTo(Object obj) zwraca:

- mniej niż 0 – gdy jej parametr jest większy niż własny obiekt
- 0 – gdy jej parametr jest równy własnemu obiektowi
- więcej niż 0 – gdy jej parametr jest mniejszy niż własny obiekt

Typy generyczne

Tradycyjne odwołania do kolekcji

```
List coll = new ArrayList();
for (Iterator iter = coll.iterator(); iter.hasNext(); )
   Person osoba = (Person) iter.next();
   System.out.println("osoba = " + person);
}
```

Wykorzystanie typów generycznych

```
List<Person> coll = new ArrayList<Person>();
for (Iterator<Person> iter = coll.iterator();
     iter.hasNext(); )
   Person osoba = iter.next();
   System.out.println("osoba = " + person);
}
```

Porównanie implementacji

Implementacje ogólnego przeznaczenia					
Kolekcje	Tablice hashujące	Tablice dynamiczne	Struktury drzewiaste	Listy	Tablica hashująca + lista
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Ćwiczenia

1. Korzystając z kolekcji, wykonaj analizator częstości występowania znaków w pliku. Zadbaj o graficzną formę (główna forma z okienkiem, w którym wczytuje się zawartość tekstu, druga forma z analizą częstości, przyciski „otwórz", „zamknij", itp.)
2. Napisz prostą i szybką książkę adresową. Elementami będą obiekty klasy Person. Zaimplementuj interfejs sortujący.
3. Sprawdź eksperymentalnie co jest szybsze: ArrayLista czy LinkedList dla następujących operacji:
    1. Wstawianie elementu na początek
    2. Wstawianie elementu do środka
    3. Wstawianie elementu na koniec
    4. Usunięcie z początku
    5. Usunięcie ze środka
    6. Usunięcie z końca
    7. Zwrócenie wartości pierwszego elementu
    8. Zwrócenie wartości środkowego elementu
    9. Zwrócenie wartości ostatniego elementu