Programowanie obiektowe

Kolekcje - pakiet „Java Collections Framework”

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

pawel.rogalinski@pwr.wroc.pl

Definicja kolekcji

Kolekcja jest obiektem, który grupuje elementy danych (inne obiekty) i pozwala traktować je jak jeden zestaw danych, umożliwiając jednocześnie wykonywanie na nim operacji np. dodawania i usuwania oraz przeglądania elementów zestawu.

Uwaga:

W innych językach programowania kolekcje są nazywane **kontenerami**. W Javie kontenery są specjalnymi „kolekcjami”, które grupują komponenty graficznego interfejsu użytkownika (GUI).

Przykłady kolekcji:

- folder poczty elektronicznej → kolekcja e-maili,
- katalog na dysku → kolekcja plików i katalogów,
- słownik ortograficzny → kolekcja poprawnych słów.

Tablice jako kolekcje

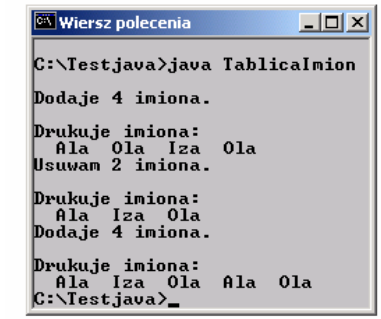
Do zapamiętywania wielu danych można wykorzystywać tablice.

Przykład:

```
class Tablica
{   private static final int ROZMIAR = 5;
    private String [] tablica = new String[ROZMIAR];
    private int ilosc = 0;

    public int size() { return ilosc; }

    public boolean add(String s)
    { if (ilosc==ROZMIAR) return false;
      tablica[ilosc++] = s;
      return true;
    }

    public boolean remove(String s)
    { for(int i=0; i<ilosc; i++)
        if( s.equals(tablica[i]) )
           { for(; i<ilosc-1; i++)
                tablica[i] = tablica[i+1];
             tablica[--ilosc] = null;
             return true;
           }
      return false;
    }

    public Object get(int i) { return tablica[i]; }
}
```


Tablice jako kolekcje cd.

```
class TablicaImion
{
    public static void main(String [] args)
    {
        Tablica imiona = new Tablica();
        System.out.println("\nDodaje 4 imiona.");
        imiona.add("Ala"); imiona.add("Ola");
        imiona.add("Iza"); imiona.add("Ola");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           { System.out.print( "  " + imiona.get(i) );
           }

        System.out.println("\nUsuwam 2 imiona.");
        imiona.remove("Ola");  imiona.remove("Ula");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           { System.out.print("  " + imiona.get(i) );
           }

        System.out.println("\nDodaje 4 imiona.");
        imiona.add("Ala");  imiona.add("Ola");
        imiona.add("Iza");  imiona.add("Ola");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           {  System.out.print("  " + imiona.get(i) );
           }
    }
}
```



Tablice jako kolekcje cd.

W języku Java tablice nie są wygodnym sposobem tworzenia kolekcji ponieważ:

- nie posiadają dedykowanych metod do obsługi kolekcji,
- rozmiar tablicy jest stały – przepełnienie tablicy może spowodować utratę pozostałych danych zapamiętanych w dedykowanych kolekcjach.

W praktycznych programach operacje na tablicach lepiej zastąpić operacjami na obiektach klasy ***Vector***.

Klasa *Vector* jako kolekcja

Klasa ***Vector*** jest ulepszoną wersją kolekcji zbudowaną w oparciu o tablicę. Posiada ona zunifikowany zestaw metod, pozwalający wykonać podobne operacje jak przy bezpośrednich odwołaniach do tablicy,

Ta kolekcja nie ma stałego, z góry definiowanego rozmiaru. Gdy wymagany jest większy rozmiar to metody kolekcji ***Vector*** automatycznie:

- tworzą nową tablicę o pożądanym rozmiarze,
- kopiują dane z dotychczasowej tablicy do nowej,
- usuwają dotychczasową tablicę.

Klasa *Vector* jako kolekcja cd.

Atrybuty:

protected int capacityIncrement
- współczynnik o który jest zwiększana wielkość tablicy,

protected int elementCount
- liczba elementów zapamiętanych w kolekcji

protected Object[] elementData
- tablica, w której są zapamiętane obiekty zawarte w kolekcji.

Konstruktory:

Vector()
- tworzy pustą tablicę o wielkości 10

Vector(int capacity)
- tworzy pustą tablicę o wielkości ***capacity***

Vector(int capacity, int increment)
- tworzy pustą tablicę o wielkości ***capacity*** i ustawia atrybut ***increment,***

Vector(Collection c)
- tworzy kolekcję wstępnie wypełnioną elementami kolekcji ***c***

Klasa *Vector* jako kolekcja cd.

Najważniejsze metody:

void add(int index, Object element)
- dodaje obiekt ***element*** na pozycji ***index***,

boolean add(Object o)
- dodaje obiekt element na końcu wektora,

boolean addAll(Collection c)
- dodaje wszystkie obiekty z kolekcji ***c***

void insertElementAt(Object obj, int index)
- wstawia element obj do tablicy na pozycji ***index***.

int capacity()
- zwraca pojemność wektora (aktualną wielkość tablicy)

int size()
-zwraca liczbę elementów pamiętanych w tablicy,

void clear()
- usuwa wszystkie elementy z tablicy

Klasa `Vector` jako kolekcja cd.

`boolean contains(Object elem)`
- sprawdza, czy element **`elem`** jest pamiętany w tablicy

`int indexOf(Object elem)`
- wyszukuje w tablicy pozycji, w której jest pamiętany element **`elem`** (porównywany za pomocą metody equals)

`elementAt(int index)`
- zwraca element pamiętany na pozycji **`index`**

`Object get(int index)`
- zwraca element pamiętany na pozycji **`index`**

`Object firstElement()`
- zwraca element pamiętany w tablicy na pozycji 0

`Object lastElement()`
- zwraca ostatni pamiętany element

`boolean isEmpty()`
- sprawdza, czy wektor jest pusty

Klasa `Vector` jako kolekcja cd.

`Object remove(int index)`
- usuwa z tablicy element pamiętany na pozycji **`index`**

`boolean remove(Object o)`
- usuwa z tablicy pierwsze wystąpienie obiektu **`o`**

`void trimToSize()`
- zmniejsza rozmiar tablicy do wielkości równej liczbie aktualnie pamiętanych elementów

Klasa `Vector` jako kolekcja - przykład

```
import java.util.*;

class WektorImion
{
    public static void main(String [] args)
    {
        Vector imiona = new Vector();

        System.out.println("\nDodaje 4 imiona.");
        imiona.add("Ala");   imiona.add("Ola");
        imiona.add("Iza");   imiona.add("Ola");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           {  System.out.print( "  " + imiona.get(i) );
           }

        System.out.println("\nUsuwam 2 imiona.");
        imiona.remove("Ola");   imiona.remove("Ula");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           {  System.out.print("  " + imiona.get(i) );
           }

        System.out.println("\nDodaje 4 imiona.");
        imiona.add("Ala");   imiona.add("Ola");
        imiona.add("Iza");   imiona.add("Ola");

        System.out.println("\nDrukuje imiona:");
        for(int i=0; i<imiona.size(); i++)
           {  System.out.print("  " + imiona.get(i) );
           }
    }
}
```

tworzenie pustego wektora

dodawanie danych do wektora

pobieranie danych z wektora

usuwanie danych z wektora

odczyt aktualnego rozmiaru wektora

Architektura kolekcji – „Collections Framework”

Architektura kolekcji – zestaw zunifikowanych struktur danych i zasad ich wykorzystywania do reprezentacji kolekcji. Składa się z:

- interfejsów,
- implementacji,
- algorytmów.

Najbardziej znane architektury kolekcji to:

- STL – *Standard Templates Library* w C++,
- klasy kolekcyjne w SmallTalk-u
- JFC – *Java Collections Framework*

Architektura kolekcji – „Collections Framework”

Interfejsy kolekcyjne – abstrakcyjne typy danych będące szablonami (wzorcami) kolekcji zawierające metody pozwalające na niezależne od szczegółów realizacyjnych operowanie na kolekcjach.

Implementacja kolekcji – konkretna realizacja metod zdefiniowanych przez interfejs kolekcyjny. Takie realizacje mogą różnić się szczegółami technicznymi, np. realizacja listy jako dynamicznej tablicy lub jako listy z dowiązaniami.

Algorytmy kolekcyjne – metody wykonujące użyteczne operacje obliczeniowe na kolekcjach, np. sortowanie i wyszukiwanie.

Podstawowe rodzaje kolekcji

Klasy JCF dostarczają środków do posługiwania się następującymi rodzajami kolekcji:

- **listy** – i ich szczególne przypadki:
  - **stosy**,
  - **kolejki**,
  - **kolejki podwójne**.
- **zbiory** i **zbiory uporządkowane**,
- **mapy** – tablice asocjacyjne (słowniki)

Podstawowe rodzaje kolekcji

Lista – zestaw elementów z których każdy znajduje się na określonej pozycji.
Elementy w liście są ułożone w takiej kolejności, w jakiej były dodawane.
Różne elementy listy mogą zawierać takie same dane.

Zbiór – zestaw niepowtarzających się elementów.
W zbiorze elementy nie mają pozycji. Nie jest możliwy dostęp do elementów zbioru „po indeksach”.
Uporządkowanie elementów w zbiorze nie zależy od kolejności dodawania/usuwania elementów.

Mapa (tablica asocjacyjna, słownik) – zestaw par *klucz* $\leftrightarrow$ *wartość*, przy czym odwzorowanie kluczy w wartość jest jednoznaczne.
Uporządkowanie par *klucz* $\leftrightarrow$ *wartość* w mapie nie zależy od kolejności dodawania/usuwania

Ogólne operacje na kolekcjach

Interfejs Collection definiuje wspólne właściwości i funkcjonalność wszystkich kolekcji (tzn. list, zbiorów i innych) poza mapami.

Podstawowe operacje na kolekcjach:

int size()
- zwraca liczbę elementów zawartych w kolekcji

boolean isEmpty()
- sprawdza, czy kolekcja jest pusta

boolean contains(Object o)
- sprawdza, czy kolekcja zawiera podany obiekt.

boolean add(Object o)
- dodaje obiekt o do kolekcji. (Uwaga: operacja opcjonalna – nie jest dozwolona dla klolekcji nimodyfikowalnych)

boolean remove(Object o)
- usuwa obiekt o z kolekcji. (Uwaga: operacja opcjonalna – nie jest dozwolona dla klolekcji nimodyfikowalnych)

Iteratory

Iterator jest obiektem służącym do przeglądania elementów kolekcji oraz ewentualnego usuwania ich przy przeglądaniu.

Każdy iterator implementuje interfejs ***Iterator***, który zawiera następujące metody:

Object next()
-zwraca kolejny element kolekcji

void remove()
-usuwa element kolekcji, zwrócony przez ostatnie odwołanie do metody ***next()***

boolean hasNext()
- zwraca ***true***, jeśli możliwe jest odwołanie do metody ***next()*** tzn. w kolekcji jest jeszcze kolejny element.

Iteratory cd.

Interfejs ***Collection*** zawiera metodę

iterator()

która zwraca iterator umożliwiający przeglądanie kolekcji.

Przykład:
Przeglądanie i ewentualne usuwanie obiektów z kolekcji ***c***

```
Iterator iter = c.iterator();

while(iter.hasNext())
   {
     Object o = iter.next();
     // ...
     // wykonanie operacji na obiekcie o
     // ...
     if ( warunek_usuniecia(o) ) iter.remove();
   }
```


Operacje grupowe na kolekcjach

Operacje grupowe na kolekcjach polegają na wykonywaniu „za jednym razem” pewnych operacji na całych kolekcjach. Należą do nich metody:

boolean addAll(Collection c)
-dodanie do dowolnej kolekcji wszystkich elementów kolekcji przekazanej przez parametr c

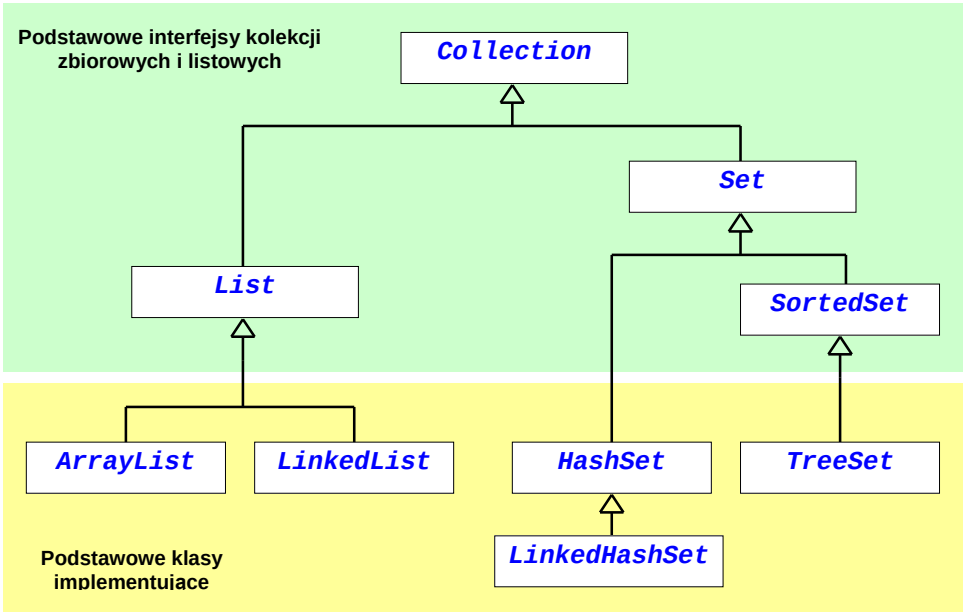
boolean removeAll(Collectin c)
- usunięcie z kolekcji wszystkich elementów, które są zawarte w kolekcji przekazanej przez parametr c

boolean retainAll(Collection)
- pozostawienie w kolekcji tylko tych elementów, które są zawarte w kolekcji przekazanej przez parametr c

void clear()
- usunięcie wszystkich elementów kolekcji.

Object [] toArray()
- zwraca tablicę obiektów zawartych w kolekcji.

Hierarchia list i zbiorów



Hierarchia map