Programowanie - Java

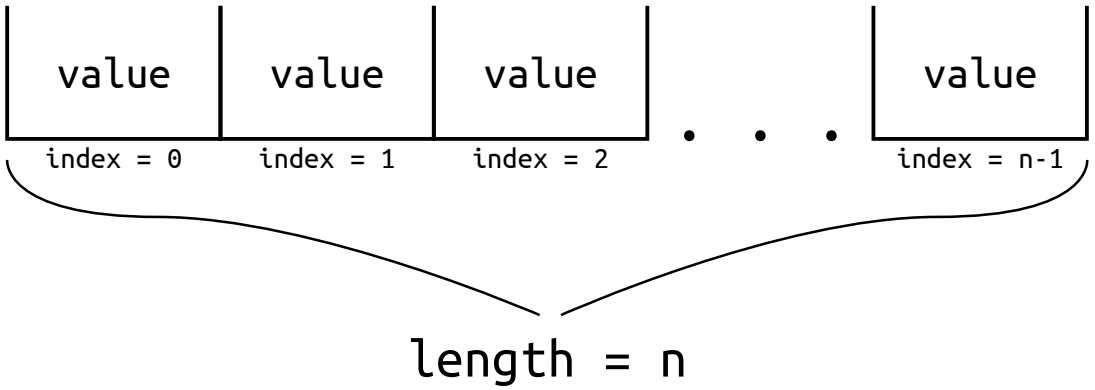
Jakub Mazurek
kuba@fenix.club

1 grudnia 2016

Spis treści

1 Struktury danych w Javie

Przez termin **struktury danych** rozumiemy zbiór różnych sposobów na organizację (uporządkowanie) informacji w programie komputerowym. Prostym przykładem struktury danych jest znana nam już **tablica**.



Rysunek 1: Tablica

Ze względu na sposób ułożenia danych w pamięci struktury danych możemy podzielić w prosty sposób na dwie grupy:

- **Przyległe (ciągłe)** (ang. *contiguous data structures*) - składają się z pojedynczego bloku w pamięci w którym elementy są ułożone jeden po drugim (w sposób ciągły). Przykładami takich struktur danych są **tablice**, **macierze** czy **tablice z haszowaniem** (ang. *hash table*).
- **Łączone** (ang. *linked data structures*) - w skład tych struktur danych wchodzą osobne fragmenty pamięci łączone ze sobą za pomocą wskaźników (ang. *pointers*). Przykładami takich struktur są **listy łączone** i **drzewa**.

Jak zobaczymy w dalszych przykładach, dobór odpowiedniej struktury danych może mieć kluczowe znaczenie w rozwiązaniu danego problemu.

Na tym etapie warto również wspomnieć, że w Javie istnieją bazowe interfejsy dla podstawowych struktur danych (np. `List`, `Set`, `Map`).

2 Listy łączone

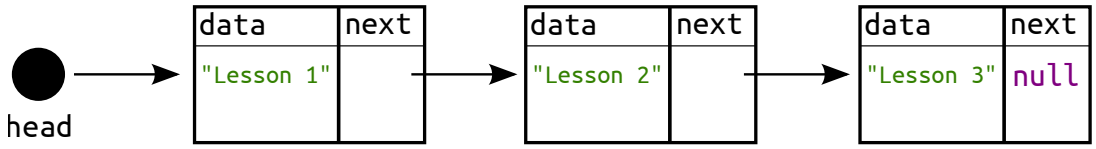
Implementacja pojedynczego elementu listy w Javie może wyglądać w następujący sposób:

```java
public class Node {

        private String data;
        private Node next;

        // getters and setters
}
```

W tym przypadku typem danych zawartym w naszej liście są ciągi znaków (obiekty typu `String`).
Lista stworzona w ten sposób będzie tzw. **listą jednokierunkową**:



Rysunek 2: Lista łączona jednokierunkowa

Do kompletnej listy brakuje nam tylko wskaźnika na jej pierwszy element (zwyczajowo nazywanego `head`). Warto również zwrócić uwagę na fakt, że wartość pola `next` dla ostatniego elementu listy będzie zawsze równa `null`.

Listy jednokierunkowe są zaimplementowane w Javie przez klasę `LinkedList`. Jest to tzw. **klasa parametryzowana**. W praktyce oznacza to, że musimy podać jej klasę obiektów, które chcemy w niej przechowywać:

```java
List<Item> items = new LinkedList<>();
```

Podstawowe operacje na listach:

- **`add(item)`** - dołącza element podany jako argument tej metody do końca listy.
- **`get(index)`** - zwraca wartość znajdującą się na podanej w argumencie pozycji w liście.
- **`contains(item)`** - zwraca `true` jeśli podany jako argument element znajduje się w liście.

2.1 Zadania

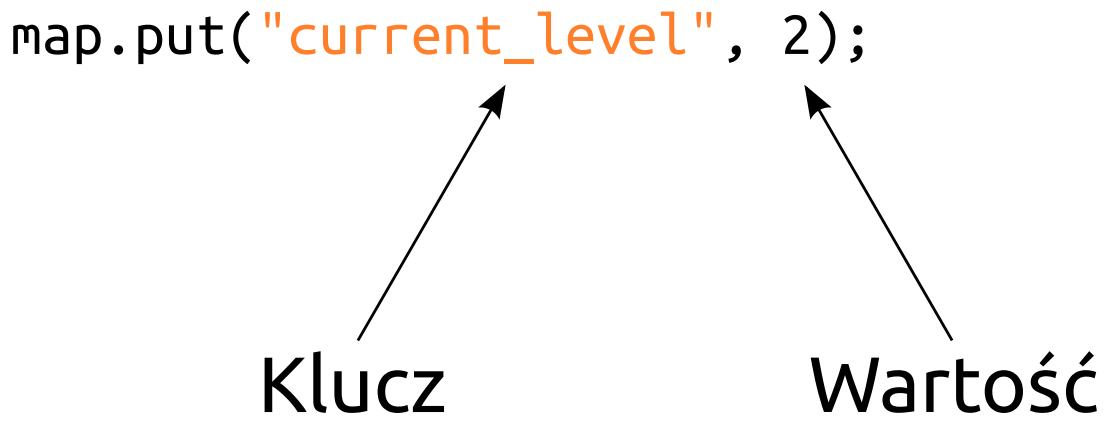
1. Stwórz nową listę łączoną (korzystając z klasy `LinkedList`) przechowującą dane type `String`.

2. Do stworzonej wcześniej listy dodaj kilka elementów w dowolnej kolejności (np. `Value1`, `Value2`, `Value3` itd.).

3. Napisz funkcję, która mając daną wartość typu `String`, sprawdzi pozycję na której znajduje się dana wartość (o ile jest ona w liście).

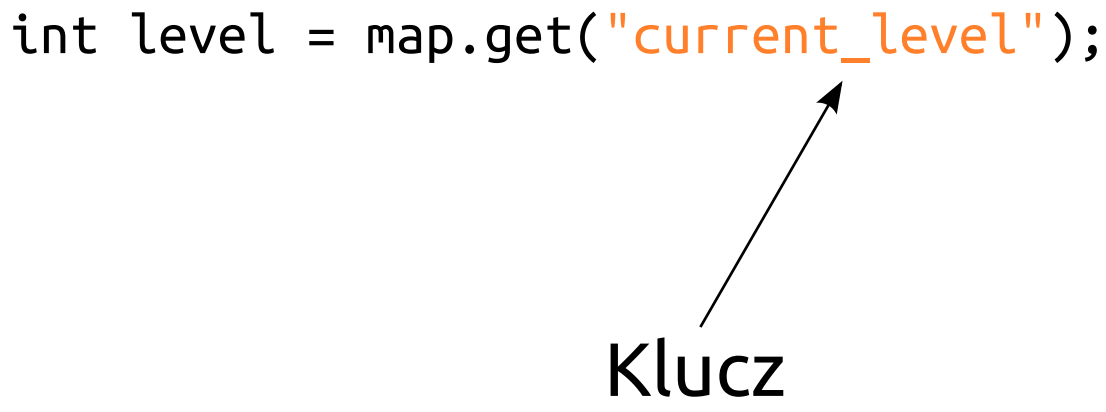
3 Mapy (tablice z haszowaniem)

Mapy służą nam do przechowywania **par klucz-wartość**. Oznacza to, że znając klucz istniejący w mapie, możemy w prosty (i szybki) sposób wyciągnąć z mapy wartość trzymaną pod podanym kluczem.



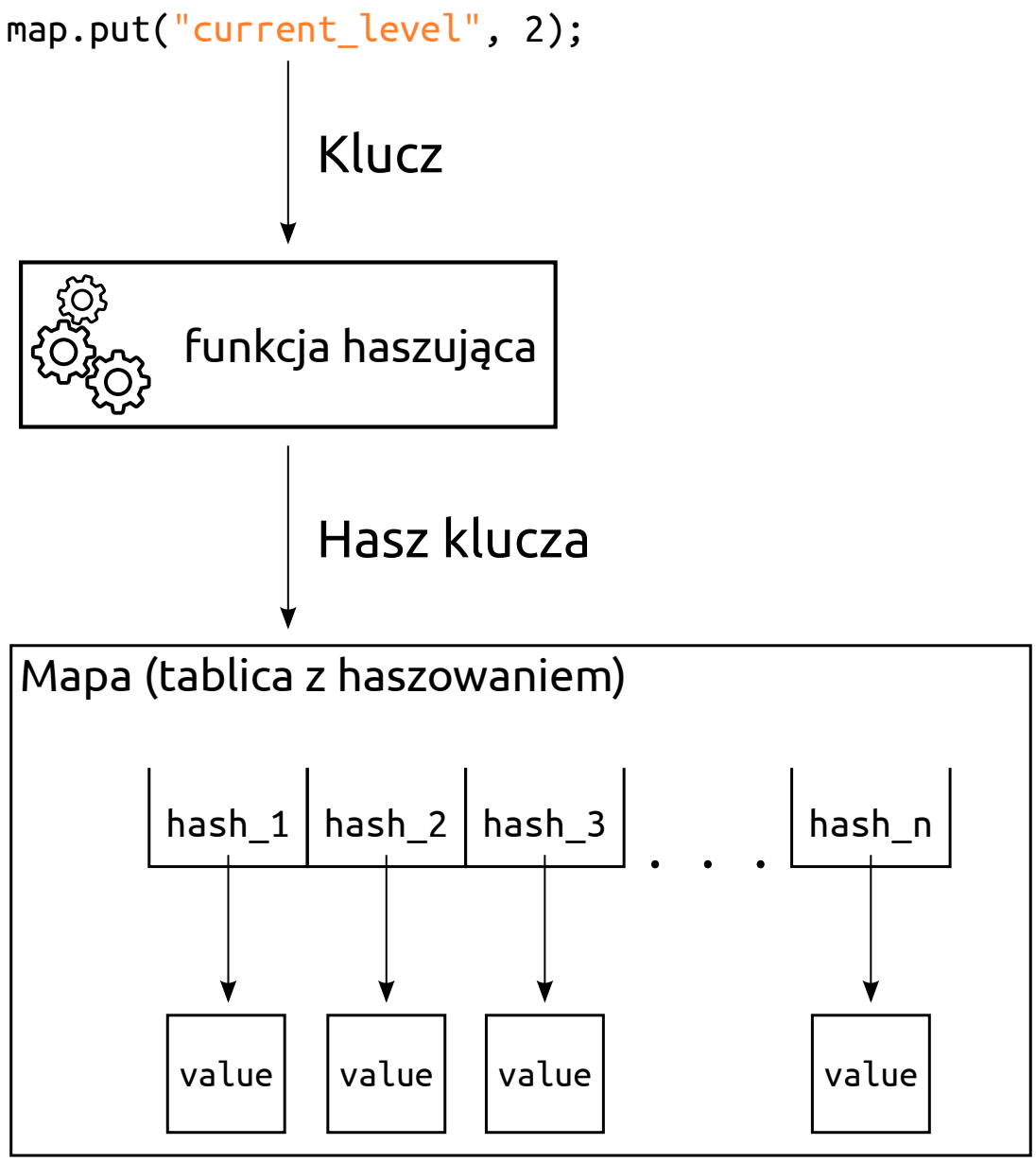
Rysunek 3: Wstawianie wartości do mapy

Znając klucz, który został użyty do wstawienia jakiejś wartości, możemy ja otrzymać w następujący sposób:



Rysunek 4: Otrzymywanie wartości z mapy

Mapy działają dzięki tzw. **funkcjom haszującym (funkcjom skrótu)** (ang. *hash functions*).



Rysunek 5: Mapa (tablica z haszowaniem)

Jednym z najprostszych przykładów zastosowania dla tej struktury danych jest implementacja słownika - dla słowa w jednym języku (klucza) mamy przypisaną wartość będącą tłumaczeniem na inny język.

Podstawowe operacje na mapach:

- **`put(key, value)`** - dodaje parę klucz-wartość do mapy (lub nadpisuje wartość, jeśli taki klucz już istnieje).
- **`get(key)`** - zwraca wartość przypisaną do klucza podanego jako argument.

- **`containsKey(key)`** - zwraca `true` jeśli mapa zawiera klucz podany jako argument tej metody.

3.1 Zadania

1. Stwórz nową mapę używając klasy `HashMap`. Twoja mapa powinna przyjmować klucze oraz wartości typu `String`.

2. Do swojej mapy dodaj kilka par klucz-wartość (np. {”Poland”, ”PL”}, {”Germany”, ”DE”} itp.).

3. Napisz funkcję, która wydrukuje do konsoli wszystkie pary klucz-wartość zawarte w Twojej mapie.