Mapy (Map)

Czego się dowiesz

- Czym są mapy,
- czym mapy różnią się od pozostałych typów kolekcji,
- kiedy warto z nich korzystać

Wstęp

Może pamiętasz, jak w lekcji wprowadzającej do kolekcji, a konkretnie w grafie prezentującym hierarchię kolekcji w collections framework, mapy stanowiły pewną osobną grupę klas, które nie implementują bezpośrednio, czy pośrednio interfejsu *Collection.*

Jest to spowodowane tym, że w odróżnieniu od list, czy zbiorów mapy służą przede wszystkim do wygodnego wyszukiwania danych po ustalonym kluczu. W mapach dane przechowywane są w postaci par **klucz** - **wartość**, gdzie oba te elementy muszą być **obiektami**. Dzięki temu możliwe jest do odwoływanie się do elementów kolekcji po kluczu niemal w taki sposób jak do elementów tablicy po indeksach. Możesz sobie więc wyobrazić to w ten sposób, że mając daną tablicę:

```
Person[] people = {new Person("Jan", "Kowalski")};
```

moglibyśmy się odwołać do obiektu "Jan Kowalski" w poniższy sposób:

```
Person jkowalski = people["Kowalski"];
```

Oczywiście mapy, tak jak wszystkie kolekcje są obiektami, więc w rzeczywistości do elementów musimy odwoływać się poprzez metody, ale zasada działania jest analogiczna jak powyższa, niestety niedziałająca w Javie tablica.

HashMap

Najczęściej wykorzystywaną spośród map jest **HashMap**. Jest ona parametryzowana dwoma parametrami **<K, V>**, gdzie K oznacza klucz (od ang. **Key**), natomiast V oznacza wartość (od ang. **Value**). Jeżeli zdecydujesz się na ten typ mapy, to wiedz, że elementy w niej przechowywane są w przypadkowej i nieposortowanej kolejności. Nie możesz się do elementów odwoływać po indeksach, ale tylko po kluczach.

Podobnie jak zbiór typu HashSet, także HashMap wymaga zaimplementowania metody hashCode() i spełnienia kontraktu z metodą equals(). Do mapy możemy dodać jeden obiekt o kluczu pustym (null) oraz wiele wartości pustych (null), ale z unikalnymi kluczami.

Najważniejsze metody w interfejsie Map to:

- **put(K key, V value)** - wstawia obiekt value typu V do zbioru i pozwala go odnaleźć pod kluczem key typu K,
- **get(K key)** - zwraca obiekt o kluczu key typu K,
- **keySet()** - zwraca zbiór wszystkich kluczy w mapie,
- **values()** - zwraca kolekcję wszystkich wartości (obiektów) przechowywanych w mapie,
- **remove(Object key)** - usuwa element mapy, dla którego klucza podanego jako parametr,
- **entrySet()** - zwraca zbiór obiektów w postaci klucz-wartość, reprezentowanych przez klasę Map.Entry<K,V>,
- **clear()** - czyści mapę ze wszystkiem przechowywanych danych.

Jako przykład zastosowania stwórzmy mapę obiektów typu Notebook, która jako klucz będzie przyjmowała model laptopa w postaci Stringa.

Notebook.java

```
import java.util.Objects;

class Notebook {
    private String producer;
    private String model;

    public Notebook(String producer, String model) {
        this.producer = producer;
        this.model = model;
    }

    public String getProducer() {
        return producer;
    }

    public void setProducer(String producer) {
        this.producer = producer;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Notebook notebook = (Notebook) o;
        return Objects.equals(producer, notebook.producer) &&
                Objects.equals(model, notebook.model);
    }

    @Override
    public int hashCode() {
        return Objects.hash(producer, model);
    }

    @Override
    public String toString() {
       return "Notebook [producer=" + producer + ", model=" + model + "]";
    }

}
```

Oprócz deklaracji pól resztę kodu oczywiście wygenerowaliśmy automatycznie. Stwórzmy klasę NotebookStore, w którym utworzymy mapę z kilkoma laptopami, a następnie wyświetlimy informacje o nich na podstawie klucza (którym będzie model laptopa) oraz wykonamy kilka operacji na danych.

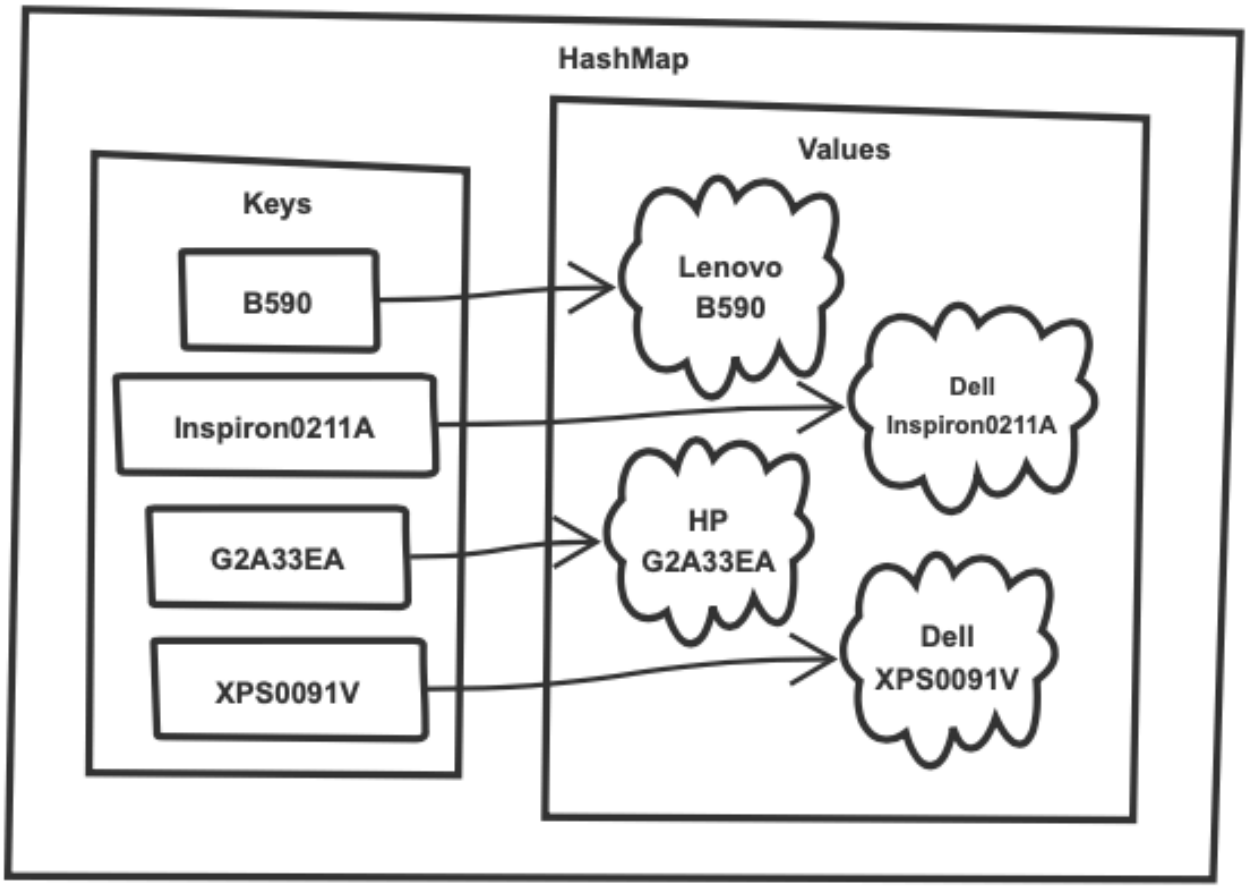
NotebookStore.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

class NotebookStore {
    public static void main(String[] args) {
        Map<String, Notebook> notebooks = new HashMap<>();
        notebooks.put("B590", new Notebook("Lenovo", "B590"));
         notebooks.put("Inspiron0211A", new Notebook("Dell", "Inspiron0211A"));
        notebooks.put("G2A33EA", new Notebook("HP", "G2A33EA"));
        notebooks.put("XPS0091V", new Notebook("Dell", "XPS0091V"));

        // wyświetlamy zbiór kluczy
        System.out.println("Modele laptopów: ");
        Set<String> keys = notebooks.keySet();
        for (String key : keys) {
            System.out.println(key);
        }

        // wyświetlamy informację o laptopach na podstawie kluczy
        String key = "G2A33EA";
        System.out.println("Znaleziono laptop o kodzie G2A33EA: ");
        Notebook foundNotebook = notebooks.get(key);
        System.out.println(foundNotebook);

        // usuwamy obiekt na podstawie klucza
        notebooks.remove("XPS0091V");

       System.out.println("Ilość produktów w sklepie: " + notebooks.size());
    }
}
```

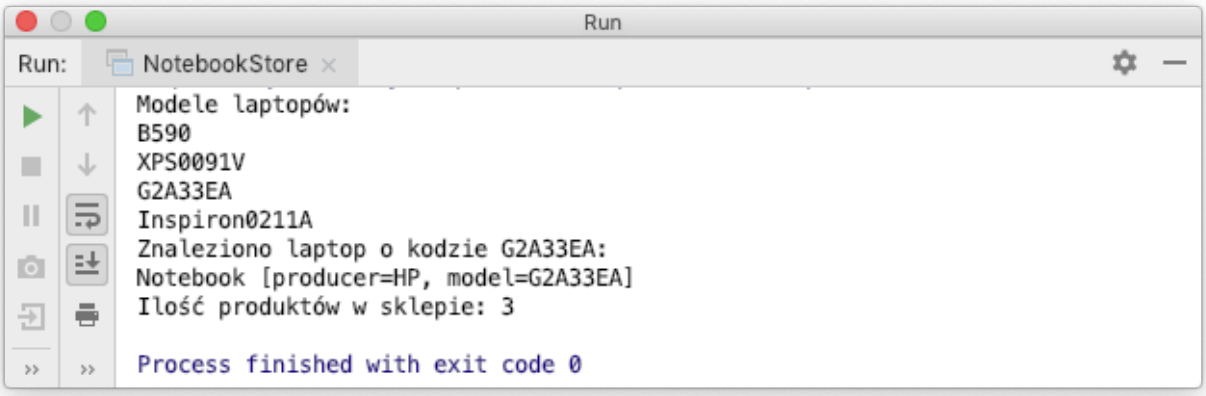
Przy tworzeniu mapy podajemy dwa parametry generyczne. W naszym przypadku mamy Map<String, Notebook>, czyli kluczami będą obiekty String, a do tych napisów będą przypisane wartości w postaci obiektów Notebook. Do mapy wartości wstawiamy korzystając z metody put(). Przyjmuje ona klucz i wartość, którą chcemy powiązać z tym kluczem, wymienione po przecinku. Metoda keySet() zwraca zbiór kluczy. Zbiór, ponieważ klucze w mapie nie mogą się powtarzać.

Metoda get() wyszukuje wartość przypisaną do danego klucza. My wyszukujemy obiekt Notebook przypisany do klucza G2A33EA i zapamiętujemy go w zmiennej foundNotebook. Jeżeli do metody get() przekażemy klucz, do którego nie jest przypisana żadna wartość, to w wyniku otrzymamy wartość null.

Możemy sobie wyobrazić, że mapa z powyższego programu wygląda w taki sposób:



Wynik działania powyższego programu:



LinkedHashMap

Podobnie jak w przypadku zbioru typu LinkedHashSet, klasa LinkedHashMap różni się od standardowej HashMap tym, że zachowuje kolejność dodawanych elementów, co może być przydatne, gdy będziemy chcieli po nich iterować.

TreeMap

Również analogicznie jak w przypadku zbioru typu TreeSet, mapa typu TreeMap różni się od standardowego HashMap tym, że obiekty są w niej sortowane zgodnie z naturalnym porządkiem **kluczy** lub zgodnie z komparatorem przekazanym w konstruktorze.

Hashtable

Klasę Hashtable można nazwać przestarzałą wersją klasy HashMap (nawet jej nazwa nie jest do końca zgodna z konwencją Javy). Klasa Hashtable jest wolniejsza, ponieważ jest klasą synchronizowaną (nie zawracaj sobie tym na razie głowy). Dodatkowo w odróżnieniu od HashMap w Hashtable nie możemy przechowywać ani wartości, ani kluczy, które są nullami. W kolejnych wersjach Javy planowane jest oznaczenie tej klasy jako @Deprecated, aby wskazać, że nie powinno się jej już używać.

Mapy i Java 9

Od Javy 9 tak samo jak w listach i zbiorach, tak również w mapach istnieją metody of(), które pozwalają tworzyć mapy na podstawie dostarczonych wartości. Ze względu na to, że obiekty w mapach przechowywane są na zasadzie par klucz-wartość, to ich tworzenie wygląda jednak nieco inaczej niż przy listach, czy zbiorach. Możemy albo przekazywać na zmianę klucze i wartości, czyli np.:

```
Map<String, Integer> people =  Map.of("Jan", 12345, "Karol", 23456, "Zofia", 34567);
```

W powyższym przykładzie w mapie powstaną trzy pary wartości: ("Jan", 12345), ("Karol", 23456), ("Zofia", 34567).

Istnieje także drugi sposób z wykorzystaniem metody ofEntries(), która może wydawać się nieco czytelniejsza ze względu na to, że nie wymieniamy na zmianę kluczy i wartości, a zamiast tego tworzymy mapę na bazie obiektów typu Entry. Powyższy przykład zapisany z użyciem tego podejścia wyglądałby tak:

```
Map<String, Integer> people = Map.ofEntries(Map.entry("Jan", 12345),
            Map.entry("Karol", 23456),
            Map.entry("Zofia", 34567));
```

Mapy utworzone z wykorzystaniem powyższych technik są niemodyfikowalne, nie możemy do nich dodawać nowych elementów, ani usuwać tych już istniejących.

Podsumowanie informacji o mapach

- Wszystkie mapy służą do przechowywania informacji w postaci <Klucz, Wartość>,
- Jeżeli zależy ci wyłącznie na szybkości działania i wygodzie indeksowania obiektów za pomocą kluczy dowolnego obiektowego typu - stosuj HashMap,
- Jeżeli potrzebny jest ci dodatkowo porządek w postaci kolejności dodawanych elementów - skorzystaj z LinkedHashMap,
- Natomiast, gdy chcesz, aby elementy w mapie były posortowane zgodnie z naturalnym porządkiem lub zdefiniowanym komparatorem - wykorzystaj klasę TreeMap.