# hashmap

### Voorbeeld

### Te gebruiken bij:

-

### Eigenschappen:

-

### Info

#### hashmap

Hier is een kort overzicht van hoe HashMap werkt:

Sleutels en waarden: Een HashMap slaat waarden op in de vorm van sleutel-waarde-paren. De sleutel is uniek en wordt gebruikt om waarden op te halen, terwijl de waarde de eigenlijke gegevens is die aan de sleutel is gekoppeld.

Geen duplicaten: Een HashMap staat geen duplicaat sleutels toe. Dit betekent dat elke sleutel slechts één keer voorkomt in de HashMap. Als je probeert een bestaande sleutel met een nieuwe waarde toe te voegen, wordt de oude waarde overschreven.

Snelle toegang: Een HashMap biedt snelle toegang tot waarden op basis van de sleutel. Dit wordt bereikt door een hashfunctie te gebruiken om de sleutel om te zetten in een hashcode, die vervolgens wordt gebruikt om de index in de interne array van de HashMap te bepalen. Hierdoor kan de HashMap efficiënt de gewenste waarde vinden zonder lineair te hoeven zoeken.

Volgorde is niet gegarandeerd: De volgorde van de elementen in een HashMap is niet gegarandeerd en kan variëren. Als je een specifieke volgorde van de elementen wilt behouden, kun je in plaats daarvan LinkedHashMap gebruiken.

java

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

// Maak een nieuwe HashMap

HashMap<String, Integer> leeftijden = new HashMap<>();

// Voeg waarden toe aan de HashMap

leeftijden.put("Alice", 25);

leeftijden.put("Bob", 30);

leeftijden.put("Charlie", 35);

// Haal de waarde op basis van de sleutel op

int leeftijdVanAlice = leeftijden.get("Alice");

System.out.println("Leeftijd van Alice: " + leeftijdVanAlice);

// Controleer of een sleutel aanwezig is

boolean heeftSleutelBob = leeftijden.containsKey("Bob");

System.out.println("Heeft sleutel Bob: " + heeftSleutelBob);

// Verwijder een waarde op basis van de sleutel

leeftijden.remove("Charlie");

// Itereer over de sleutel-waarde-paren in de HashMap

for (String naam : leeftijden.keySet()) {

int leeftijd = leeftijden.get(naam);

System.out.println(naam + ": " + leeftijd);

}

}

}

In dit voorbeeld wordt een HashMap met de naam "leeftijden" gemaakt om de leeftijden van verschillende personen op te slaan. Sleutels worden vertegenwoordigd door Strings en waarden worden vertegenwoordigd door Integers. Waarden worden toegevoegd met de "put()" methode,

#### Set

Naast Lists en Maps heb je dan ook nog... Sets! Je kunt een Set gebruiken wanneer je absoluut geen dubbele waardes in de verzameling wil hebben. Denk bijvoorbeeld aan een lijst met gebruikersnamen voor een social media platform: je wil geen twee gebruikers met dezelfde gebruikersnaam hebben, anders weten we niet meer wie wie is. In zo'n geval is het handig om een Set te gebruiken - en dan specifiek de implementatie HashSet:

Set<String> gebruikersnamen = new HashSet<>();

In onderstaand voorbeeld kun je zien hoe we proberen om twee keer dezelfde gebruiker (novi\_de\_beste) toe te voegen aan onze set. Als we echter alle gebruikersnamen uit de Set printen, zien we slechts één keer "novi\_de\_beste" in de output staan. Is je al iets opgevallen aan de volgorde van de lijst...?

public class Main {

public static void main(String[] args) {

Set<String> gebruikersnamen = new HashSet<>();

gebruikersnamen.add("novi\_de\_beste");

gebruikersnamen.add("master1337");

// etc.

gebruikersnamen.add("novi\_de\_beste");

gebruikersnamen.add("BuurmanBob");

for(String gebruikersnaam : gebruikersnamen){

System.out.println(gebruikersnaam);

}

}

}

Het mooie van het gebruik van een Set, is dat er alleen unieke waardes in kunnen staan. Het nadeel is echter dat er geen volgorde wordt aangehouden, dus waardes worden niet opgeslagen in de volgorde dat ze zijn toegevoegd. Zodoende is het ook niet mogelijk om een get()-methode aan te roepen op een Set. Om waardes uit te lezen, zul je altijd een for-loop moeten gebruiken zoals hierboven gedemonstreerd is.

HashSet is een klasse in Java die de implementatie is van de Set-interface. Het is een gegevensstructuur die elementen opslaat in willekeurige volgorde zonder duplicaten toe te staan. Het maakt gebruik van een hashfunctie om de elementen intern op te slaan en te organiseren.

Hier zijn enkele belangrijke kenmerken van HashSet:

1. Unieke elementen: HashSet staat geen duplicaten toe. Als je probeert een duplicaat toe te voegen aan een HashSet, wordt het genegeerd en blijft alleen het oorspronkelijke element behouden.
2. Geen volgorde: HashSet slaat elementen op in willekeurige volgorde. Er is geen garantie dat de volgorde waarin elementen zijn toegevoegd, behouden blijft. Als je de elementen in een specifieke volgorde wilt verkrijgen, moet je een andere implementatie, zoals LinkedHashSet of TreeSet, overwegen.
3. Snel opzoeken: HashSet maakt gebruik van hashcodes om elementen intern op te slaan en te organiseren. Hierdoor is het opzoeken van elementen in een HashSet zeer efficiënt, zelfs bij grote sets.
4. Geen duplicaten: HashSet gebruikt de equals() en hashCode() methoden van de elementen om duplicaten te identificeren. Daarom moet je ervoor zorgen dat de elementen van je HashSet de equals() en hashCode() methoden correct implementeren om duplicaten correct te kunnen identificeren.

Hier is een voorbeeld van het gebruik van HashSet:

Java

HashSet<String> set = new HashSet<>();

set.add("Apple");

set.add("Banana");

set.add("Orange");

set.add("Apple"); // Dit wordt genegeerd omdat "Apple" al aanwezig is

System.out.println(set.size()); // Geeft 3 weer, omdat duplicaten zijn genegeerd

set.remove("Banana"); // Verwijder "Banana" uit de set

System.out.println(set.contains("Orange")); // Controleer of "Orange" aanwezig is (true)

for (String element : set) {

System.out.println(element); // Itereer over de elementen en print ze

}

In dit voorbeeld wordt een HashSet aangemaakt en worden er enkele elementen aan toegevoegd. Het toevoegen van een duplicaat wordt genegeerd en de grootte van de set wordt bepaald met size(). Elementen kunnen worden verwijderd met remove(), en je kunt controleren of een element aanwezig is met contains(). Je kunt ook itereren over de elementen van de set met behulp van een for-each-lus.

contains(Object element): Deze methode controleert of een bepaald element aanwezig is in de HashSet. Het retourneert true als het element gevonden wordt, anders retourneert het false.

1. Iteratie: Je kunt de elementen in een HashSet itereren met behulp van een for-each-lus of een iterator. Hierdoor kun je alle elementen in de set doorlopen en de gewenste gegevens vinden of gebruiken.

java

Copy code

HashSet<String> set = new HashSet<>();

set.add("Apple");

set.add("Banana");

set.add("Orange");

boolean containsBanana = set.contains("Banana");

System.out.println("Bevat Banana: " + containsBanana); // Geeft true weer

// Iteratie met een for-each-lus

for (String element : set) {

if (element.equals("Orange")) {

System.out.println("Gevonden: " + element);

break;

}

}

// Iteratie met een iterator

Iterator<String> iterator = set.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

if (element.equals("Apple")) {

System.out.println("Gevonden: " + element);

break;

}

}

In dit voorbeeld wordt gecontroleerd of de HashSet het element "Banana" bevat met behulp van contains(). Vervolgens wordt er door de set geïtereerd met behulp van een for-each-lus en een iterator om te zoeken naar de elementen "Orange" en "Apple". Zodra het gewenste element is gevonden, kan je de benodigde acties ondernemen.

Let op: de HashSet geen garantie biedt voor een specifieke volgorde van elementen, dus je moet niet vertrouwen op de volgorde van iteratie om specifieke elementen te vinden.