Chapter 01 HashMaps

HashMap is a collection that stores elements in key/value pairs. Here, keys are unique identifiers used to associate each value on a map.

1. Creating a HashMap

In order to create a hash map, we must import the `java.util.HashMap` package first. Once we import the package, here is how we can create Hashmaps in Java.

Syntax:

```
HashMap<keyType, valueType> numbers = new HashMap<>();
```

For example:

```
HashMap<String, Integer> numbers = new HashMap<>();
```

In the above example, we have created a HashMap named `numbers`. Here, the type of keys is `String` and the type of values is `Integer`. Note that we used Integer not `int`. It is because HashMaps store references.

2. HashMap Methods

The `HashMap` class provides various methods to perform different operations on HashMaps.

Method	*Description*
`put()`	inserts the specified element (key/value mapping) to the HashMap.
`putAll()`	inserts all the key/value mappings from the specified Map to the HashMap.
`get()`	returns the value corresponding to the specified key in the HashMap.
`getOrDefault()`	returns the specified default value if the element for the specified key is not found in the HashMap.
`keySet()`	returns a set view of all the keys present in entries of the HashMap.
`values()`	returns a view of all the values present in entries of the HashMap.
`replace()`	replaces the element for the specified key with the specified new value in a HashMap
`replaceAll()`	replaces all elements of the HashMap with the result from the specified function.
`remove()`	returns the value associated with the key and removes the entry
`clear()`	removes all elements from the HashMap
`merge()`	merges the specified elements to the HashMap
`clone()`	makes the copy of the HashMap
`size()`	returns the number of elements in HashMap

`containsKey()`	checks if the specified key is present in HashMap
`containsValue()`	checks if HashMap contains the specified value
`isEmpty()`	checks if the HashMap is empty

3. HashMap Initialization

3.1 Initialization with `put()` method

We create a HashMap and use `put()` method to add elements to the HashMap. For example:

```
HashMap<String, String> hashMap = new HashMap<>();
hashMap.put("key1", "value1");
hashMap.put("key2", "value2");
hashMap.put("key3", "value3");
```

We can also initialize the map using the double-brace syntax. For example:

```
HashMap<String, String> hashMap = new HashMap<>(){{
    put("key1", "value1");
    put("key2", "value2");
    put("key3", "value3");
}};
```

3.2 Initialization using Another HashMap

We can initialize an HashMap using another Map:

```
HashMap<String, Integer> hashMap = new HashMap<>(anotherHashMap);
```

4. Basic Operations on HashMaps

The HashMap class provides various methods to perform different operations on HashMaps. We will look at some commonly used Hashmaps operations such as:

- Add elements
- Access elements
- Update elements
- Remove elements

4.1 Adding Elements

To add a single element to a HashMap, we use the `put()` method of the HashMap class.

Example 01: Use `put()` method to add elements to a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        // Create a hashmap
        HashMap<String, Integer> scores = new HashMap<>();

        // Add elements to hashmap
        scores.put("Math", 90);
        scores.put("Physics", 100);
        scores.put("Chemistry", 95);

        System.out.print("HashMap: " + scores);
    }
}
```

Output:

```
HashMap: {Chemistry=95, Math=90, Physics=100}
```

And, when the HashMap contains the mapping for the specified key, then the `put()` method replaces the value associated with the specified key.

Example 02:

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        // Create a hashmap
        HashMap<String, Integer> scores = new HashMap<>();

        // Add elements to hashmap
        scores.put("Math", 90);
        scores.put("Physics", 100);
        scores.put("Math", 85);

        System.out.print(scores);
    }
}
```

Output:

```
HashMap: {Math=85, Physics=100}
```

To add all the elements from a HashMap to another, we can use the `putAll()` method.

Example 03: In this example, we will create two Hashmaps named `hashMap1` and `hashMap2`. Then, we add all the elements from `hashMap2` to `hashMap1`.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        // create an HashMap
        HashMap<String, Integer> hashMap1 = new HashMap<>();
        hashMap1.put("Two", 2);
        hashMap1.put("Three", 3);

        // create another HashMap
        HashMap<String, Integer> hashMap2 = new HashMap<>();
        hashMap2.put("One", 1);
        hashMap2.put("Two", 22);

        // Add all elements from hashMap2 to hashMap1
        hashMap1.putAll(hashMap2);

        System.out.println("Numbers: " + hashMap1);
    }
}
```

Output:

```
{One=1, Two=22, Three=3}
```

Notice that the value for the key `"Two"` is changed from `22` to `2`. It is because key `"Two"` is already present in the `hashMap1`. Hence, the value is replaced by the new value from `hashMap2`.

4.2 Accessing Elements

The `get()` method takes the key as its argument and returns the corresponding value associated with the key.

Example 04: Use the `get()` method to access the value from a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args){
        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "C++");

        String value = languages.get(2);

        System.out.println("Value at key 2: " + value);
    }
}
```

Output:

```
Value at key 2: Python
```

We can use the `getOrDefault ()` method to return the specified default value if the specified key is not found in the hashmap.

Example 05: Use the `getOrDefault()` method to access the value from a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args){
        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "C++");

        String value = languages.getOrDefault(5, "Key Not Found");

        System.out.println("Value at key 5: " + value);
    }
}
```

Output:

```
Value at key 5: Key Not Found
```

We can also access the keys and values of a HashMap as set views using `keySet()` and `values()` methods respectively.

Example 06: Using `keySet()` and `values()` methods.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args){
        HashMap<Integer, String> languages = new HashMap<>();

        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "JavaScript");

        // return set view of keys using keySet()
        System.out.println("Keys: " + languages.keySet());

        // return set view of values using values()
        System.out.println("Values: " + languages.values());
    }
}
```

Output:

```
Keys: [1, 2, 3]
Values: [Java, Python, JavaScript]
```

4.3 Updating Elements

We can use the `replace`() method to change the value associated with a key in a HashMap.

Example 07: Replace a mapping in a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args){
        // create an HashMap
        HashMap<Integer, String> languages = new HashMap<>();

        // add entries to HashMap
        languages.put(1, "Python");
        languages.put(2, "English");
        languages.put(3, "JavaScript");
        System.out.println("HashMap: " + languages);

        // replace mapping for key 2
        String value = languages.replace(2, "Java");

        System.out.println("Replaced Value: " + value);
        System.out.println("Updated HashMap: " + languages);
    }
}
```

Output:

```
Replaced Value: English
Updated HashMap: {1=Python, 2=Java, 3=JavaScript}
```

If the HashMap does not contain any mapping for the specified key, then the `replace`() method returns `null`.

`HashMap` class also provides a method called `replaceAll()` that replaces each value with the result of a specified function.

Example 08: Change all values in a HashMap to uppercase using the `replaceAll()` method.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "java");
        languages.put(2, "javascript");
        languages.put(3, "python");

        // Change all value to uppercase
        languages.replaceAll((key, value) -> value.toUpperCase());

        System.out.println(languages);
    }
}
```

Output:

```
{1=JAVA, 2=JAVASCRIPT, 3=PYTHON}
```

Here,

- `(key, value) -> value.toUpperCase()` is a lambda expression. It converts all values of the HashMap into uppercase and returns it.
- `replaceAll()` replaces all values of the HashMap with values returned by the lambda expression.

Example 09: Replace all values in a HashMap with the square of keys.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, Integer> numbers = new HashMap<>();
        numbers.put(5, 0);
        numbers.put(8, 1);
        numbers.put(9, 2);

        // replace all value with the square of key
        numbers.replaceAll((key, value) -> key * key);

        System.out.println(numbers);
    }
}
```

Output:

```
{5=25, 8=64, 9=81}
```

Here,

- `(key, value) -> key * key` computes the square of key and returns it
- `replaceAll()` replaces all values of the HashMap with values returned by the lambda expression `(key, value) -> key * key`

4.4 Removing Elements

To remove elements from a hashmap, we can use the `remove()` method.

Example 09:

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "JavaScript");

        // remove element associated with key 2
        String value = languages.remove(2);
        System.out.println("Removed value: " + value);
        System.out.println("Updated HashMap: " + languages);
    }
}
```

Output:

```
Removed value: Python
Updated HashMap: {1=Java, 3=JavaScript}
```

We can also remove the entry only under certain conditions. For example,

```
remove(2, "C++");
```

Here, the `remove()` method only removes the entry if the key `2` is associated with the value `"C++"`. Since `2` is not associated with `"C++"`, it does not remove the entry.

4.5 Iterating Through a HashMap

To iterate through each entry of a HashMap, we can use Java for-each loop. We can iterate through keys only, vales only, and key/value mapping.

Example 10: Iterate through a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        // create a HashMap
        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "JavaScript");

        // iterate through keys only
        System.out.print("Keys: ");
        for (Integer key : languages.keySet()) {
            System.out.print(key + "   ");
        }

        // iterate through values only
        System.out.print("\nValues: ");
        for (String value : languages.values()) {
            System.out.print(value + "   ");
        }
    }
}
```

Output:

```
Keys: 1   2   3
Values: Java   Python   JavaScript
```

5. Nested HashMaps

A nested HashMap is Map inside a Map, i.e., the value of the nested HashMap is another HashMap.

5.1 Creating Nested HashMaps

To create a nested map, we will first have to create a normal HashMap, just the value of this map will be another HashMap. For example,

```
HashMap<String, HashMap<Integer, String>> nestedMap = new HashMap<>();
```

A table below shows how a nested HashMap can store data. nested HashMap are very useful when there is nested information. For example, in the table below, each person has an identifier "name" and has several pieces of nested information (age, heigh and weight).

Name	Information
John	Age: 50 Height: 170 Weight: 70
Mary	Age: 35 Height: 150 Weight: 50

Example 11: Create a nested HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<String, Integer> person1 = new HashMap<>();
        person1.put("Age", 50);
        person1.put("Height", 170);
        person1.put("Weight", 70);

        HashMap<String, Integer> person2 = new HashMap<>();
        person2.put("Age", 35);
        person2.put("Height", 150);
        person2.put("Weight", 50);

        HashMap<String, HashMap<String, Integer>> person_List = new HashMap<>();
        person_List.put("John", person1);
        person_List.put("Mary", person2);

        System.out.println(person_List);
    }
}
```

Output:

```
{John={Height=170, Age=50, Weight=70}, Mary={Height=150, Age=35, Weight=50}}
```

5.2 Adding Element to Nested Map

We can add new elements to an existing nested HashMap by first retrieving the nested HashMap from the outer HashMap using the `get()` method, and then using the `put()` method to add the new elements to the inner HashMap.

Example 12: Add new elements to a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, String> hashMap1 = new HashMap<>();
        hashMap1.put(1, "Java");
        hashMap1.put(2, "Python");

        HashMap<Integer, String> hashMap2 = new HashMap<>();
        hashMap2.put(1, "C++");
        hashMap2.put(2, "JavaScript");

        HashMap<Integer, HashMap<Integer, String>> hashMap2D = new HashMap<>();
        hashMap2D.put(1, hashMap1);
        hashMap2D.put(2, hashMap2);

        //Adding another element for key 1
        hashMap2D.get(1).put(3, "SQL");

        System.out.println(hashMap2D);
    }
}
```

Output:

```
{1={1=Java, 2=Python, 3=SQL}, 2={1=C++, 2=JavaScript}}
```

5.3 Removing Element from Nested Map

To remove elements from nested HashMap, invoke remove() similar to the previous example.

Example 13: Remove elements from a HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, String> hashMap1 = new HashMap<>();
        hashMap1.put(1, "Java");
        hashMap1.put(2, "Python");

        HashMap<Integer, String> hashMap2 = new HashMap<>();
        hashMap2.put(1, "C++");
        hashMap2.put(2, "JavaScript");

        HashMap<Integer, HashMap<Integer, String>> hashMap2D = new HashMap<>();
        hashMap2D.put(1, hashMap1);
        hashMap2D.put(2, hashMap2);

        //Remove an element at key 1
        hashMap2D.get(1).remove(2);

        System.out.println(hashMap2D);
    }
}
```

Output:

```
{1={1=Java}, 2={1=C++, 2=JavaScript}}
```

5.4 Iterating over Nested HashMap

We can iterate over a nested map just like we will iterate over a normal HashMap.

Example 14: Iterate over a nested HashMap.

```
import java.util.HashMap;

class Example {
    public static void main(String[] args) {
        HashMap<Integer, String> hashMap1 = new HashMap<>();
        hashMap1.put(1, "Java");
        hashMap1.put(2, "Python");

        HashMap<Integer, String> hashMap2 = new HashMap<>();
        hashMap2.put(1, "C++");
        hashMap2.put(2, "JavaScript");

        HashMap<Integer, HashMap<Integer, String>> hashMap2D = new HashMap<>();
        hashMap2D.put(1, hashMap1);
        hashMap2D.put(2, hashMap2);

        System.out.println(hashMap2D);

        // iterate through values only
        System.out.print("\nValues: ");
        for (HashMap<Integer, String> map : hashMap2D.values()) {
            for (String value : map.values()) {
                System.out.print(value + "   ");
            }
        }
    }
}
```

Output:

```
Values: Java   Python   C++   JavaScript
```

Exercises

Write the programs below using HashMaps; you are **NOT** allowed to use standard arrays or ArrayLists.

1. *(Guess the capitals)* Write a program that repeatedly asks the user to enter a capital city for a country. Upon receiving the user input, the program reports whether the answer is correct. Assume that 10 countries and their capital cities are stored in a HashMap. The program asks the user to answer all the countries' capital cities and displays the total correct count. The user's answer is not case-sensitive. And the order of the questions is randomly (in this case, you are allowed to use arrays). Implement the program using a HashMap to represent the data in the following table:

Cambodia	Phnom Penh
Thailand	Bangkok
China	Beijing
Japan	Tokyo
India	Delhi
Malaysia	Kuala Lumpur
…	…

   Here is a sample run:

```
What is the capital of Alabama? Montogomery ↵ Enter
The correct answer should be Montgomery
What is the capital of Alaska? Juneau ↵ Enter
Your answer is correct
What is the capital of Arizona? ...
...
The correct count is 35
```

2. *(Compute the weekly hours for each employee)* A table below shows the seven-day work hours of eight employees. Write a program that displays employees in ascending order of the total hours.

	Su	M	T	W	Th	F	Sa
Employee 0	2	4	3	4	5	8	8
Employee 1	7	3	4	3	3	4	4
Employee 2	3	3	4	3	3	2	2
Employee 3	9	3	4	7	3	4	1
Employee 4	3	5	4	3	6	3	8
Employee 5	3	4	4	6	3	4	4
Employee 6	3	7	4	8	3	8	4
Employee 7	6	3	5	9	2	7	9

3. Store id, name, and department of three employees in a HashMap. Then, write a program that includes a menu that will allow user to select any of the following features:
   a. Display all employees
   b. Add a new employee
   c. Delete employee by id
   d. Update employee by id
   e. Search employee by id
   f. Exit the program

When display the employee(s), the data should be arranged in a tabular format, for example:

```
-------------------------------------------------------------------------------
ID          Name            Salary          Department
-------------------------------------------------------------------------------
1           Chan Dara       800             ITE
2           Sok Sophea      900             BioE
3           Keo Tola        500             TEE
-------------------------------------------------------------------------------
```

4. Write an ATM machine program. First, stores the account_no (001, 002, …005), name, balance and password of five customers in a HashMap. When the program starts, it asks the user to enter the account_no and password to login. If login succeeded, it displays the following menus:
   a. Balance
   b. Withdraw
   c. Deposit
   d. Transfer
   e. Exit the program

   Note:
   - If the login is not successful, repeat it again; do not end the program there.
   - If the balance is not enough for withdrawing, ask the user if they want to enter another amount.
   - Before transferring, ask the user to enter their password again to confirm it. Let the user know if the transaction is successful or not (the user might have entered the wrong password when confirming, the balance was not enough, or the receiver's account_no was not found).

Reference

[1] Y. Daniel Liang. 'Introduction to Java Programming', 11e – 2019

[2] https://www.programiz.com/java-programming