HashMap Class

HashMap holds series of key-value pairs. Where every key is can map to at most one value. In HashMap the keys are unique (no duplicates allowed). However, there is no such restriction on values thus two keys can both point to same value. Let's understand some real cases where working with key-value pairs might help:

Some Example Scenarios of Key-Value pairs:

1. **Telephone Directory:** A telephone directory is a really simple example for this. Telephone directory is nothing but a key-value pair of name and numbers. Here we have a number as key (because they are unique) and name as value. So each pair in a telephone directory resembles telephone number to name mapping (a kind of key to value mapping). All the phone numbers are unique i.e. there cannot be two entries of the same telephone number. Values here denote names of people owning these numbers. Now we know that we can have multiple people with the same names and also a person can have multiple numbers. So uniqueness isn't required with names. So we can have two phone numbers pointing to the same name.
2. **Login Credentials:** A login system accepts username and password to authenticate. These also can be modelled as key-value pairs. Here key is username which has to be unique. And value is the password. So a new user will always need a unique username (a new username for every new account). Passwords on the other hand can be same against two usernames (however, we don't advise this). *Please note a typical login system in production isn't this simple. But, follows this same general idea.*

Use HashMap in following situations:

- We need to model key-value relationship.
- We need fast search on keys. And want to store values against each key. A HashSet obviously isn't fit for this scenario.

Syntax:

```
HashMap<KeyType, ValueType> map_name = new HashMap<KeyType, ValueType>();
//Where Type = Any Valid ClassName
```

Like other Collection Framework classes HashMap too can only hold objects. So if you want to store any primitive data type then you have to use their corresponding wrapper class.

Class Name for different data types:

- **Integer** class for int
- **String** class for String
- **Character** class for char

Example 1: Creating Simple HashMap

```
1  HashMap<Integer, Integer> map1 = new HashMap<Integer, Integer>();
2  HashMap<String, Integer> map2 = new HashMap<String, Integer>();
```

Explanation:

map1 have both key and value of Integer data type. map2 has key of type String and value of type Integer.

Basic Methods: put() and size()

Some basic operations:

- put(key, value) : adds the given <key, value> pair in the map. If key already exist then new value overwrites the old one.
- size() : returns the size of map (total key-value pairs present)

Example 2: Using simple methods

```
1  HashMap<String, String> phoneDirectory = new HashMap<String, String>();
2
3  //adding key-value pair to our map
4  phoneDirectory.put("9999999911", "Mohit");
5  phoneDirectory.put("9999999915", "Akash");
6
7  int len = phoneDirectory.size();          //getting current size of our map
8  System.out.println(len);
9  System.out.println(phoneDirectory);       //printing our map
10
11 //notice we are using existing key
12 phoneDirectory.put("9999999911", "Tushar");
13 len = phoneDirectory.size();
14 System.out.println(len);
15
16 System.out.println(phoneDirectory);
```

Output:

```
2
{9999999911=Mohit, 9999999915=Akash}
2
{9999999911=Tushar, 9999999915=Akash}
```

Explanation:

Here we have both key and value as String. Key is our telephone number and values are the name mapped to each telephone number. Now as you can see above when adding a new entry with the existing key the size remains same. So put() method here instead of adding a new key just updates the record (corresponding to the given key). This ensures no duplicity.

Problem 1	Give output of following code.

```
1  HashMap<Integer, String> adhaarCardHolders = new HashMap<Integer, String>();
2
3  adhaarCardHolders.put(1234, "mukesh.kumar@gmail.com");
4  adhaarCardHolders.put(1276,"rana.209@yahoo.com");
5  adhaarCardHolders.put(1234,"kumarmukesh@gmail.com");
6  adhaarCardHolders.put(2439,"rohit.official@gmail.com");
7  adhaarCardHolders.put(1276,"rana.208@yahoo.co.in");
8  len = adhaarCardHolders.size();
9  System.out.println(len);
10
11 System.out.println(adhaarCardHolders);
```

Getting certain value and Checking presence of a key

Checking if a certain element is part of given set is simple.

- get(key) : returns corresponding value if the key is present, otherwise returns null.
- containsKey(key) : returns true if the key is present in map.

Example 3.1: Getting values and checking keys

```
1  // Assume HashMap : users<String, String> contains
2  // => {"nehakr"="123", "mukesh70"="ab#d", "akshay.cool"="Ab$2"}
3
4  //given below is the function to authenticate user
5
6  public void auth(String username, String password, HashMap<String, String> users)
7  {
8     if(users.containsKey(username))
9     {
10          System.out.println("Username found");
11
12          String correctPassword = users.get(username);
13
14          if(password.equals(correctPassword))
15          {
16                System.out.println("Login Successful");
17          }
18          else
19          {
20                System.out.println("Password incorrect");
21          }
22    }
23    else
24    {
25          System.out.println("Username does not exist");
26    }
27 }
```

Input	Output
auth("mukesh70", "ab#d", users)	Username found Login Successful
auth("ankita.kapoor", "kprankt", users)	Username does not exist
auth("nehakr", "133", users)	Username found Password incorrect

Explanation:

The above program models a simple login system. All the users are stored in a HashMap (users) with username as keys and password as values. The program checks if the entered username exist in our map or not. If it's not present then an error message is displayed. If it's present then we proceed to compare the associated password with this username.

Example 3.2: Printing count of each String in the given String array

```
1  String[] strArr = {"ab", "ba", "ab", "aba", "ba"};
2
3  HashMap<String, Integer> myMap = new HashMap<String, Integer>();
4
5  for(int i = 0; i < strArr.length; i++)
6  {
7     String str = strArr[i];
8
9     if(myMap.containsKey(str))             //if key(String) already present
10    {                                      //
11          int count = myMap.get(str);      //then
12          count = count + 1;               //increment frequency
13          myMap.put(str, count);           //update frequency against key
14    }
15    else
16    {
17          myMap.put(str, 1);               //adding key(String) for the first time
18    }
19 }
20 int len = myMap.size();                   //total unique Strings
21 System.out.println(len);
22
23 System.out.println(myMap);                //printing our map
```

Output:

```
3
{ab=2, ba=2, aba=1}
```

Explanation:

The above program creates a map of unique Strings from given array. It then keeps count of each String in a map. In each iteration we pick a String from the array and check if the map already contains it or not. If it's not present in map then we add an entry and update the corresponding value by 1. If it's

already present then we just increment the value against this String by 1. At the end of the loop we'll have a map with all unique Strings as key and their frequency in given array as corresponding value.

Problem 2 | **Refer the function given below and give output of code that follows.**

Function

```
1  // Assume HashMap : users<String, String> contains
2  // => {"nehakr"="123", "mukesh70"="ab#d"}
3
4  public void update(String usrname, String pass, HashMap<String, String> users)
5  {
6      if(users.containsKey(usrname))
7      {
8             System.out.println("Username exists");
9             users.put(usrname, pass);
10     }
11     else
12     {
13            users.put(usrname, pass);
14     }
15
16     System.out.println(users);
17 }
```

Code

```
1  update("mukesh70", "ab#d1", users);
2  update("nehakr", "123", users);
3  update("neeraj19", "1#2", users);
```

Problem 3 | **Give output of following code.**

```
1  int[] arr = {10, 20, 10, 30, 60, 90, 50, 10, 100, 150, 110};
2
3  HashMap<String, Integer> filter = new HashMap<String, Integer>();
4  filter.put("0-50", 0);
5  filter.put("51-100", 0);
6
7  for(int i = 0; i < arr.length; i++)
8  {
9      int num = arr[i];
10
11     if(num<=50)
12     {
13            int count = filter.get("0-50");
14            count = count + 1;
15            filter.put("0-50", count);
16     }
17     else if(num <= 100)
18     {
19            int count = filter.get("51-100");
```

```
20            count = count + 1;
21            filter.put("51-100", count);
22        }
23    }
24
25    System.out.println(filter);
```

Key Points

HashMap is part of Java Collection Framework. Some of the key points about HashMap are:

- HashMap class implements Map interface. Map interface provides basic methods like put(), get(), containsKey(), etc.
- HashMap contains elements in key value pairs. E.g. <Key1, Value1>, <Key2, Value2>…..
- Each key can map to at most one value.
- Key must be unique but the values can be duplicate. So we cannot have duplicate keys. But, two different keys can have same (duplicate) values.
- Searching for a key is very fast in HashMap.

Problem 4 **AnalyzeMarks**	**Complete the function given below. The function takes an integer arrays containing marks of students as input. The function then prints the count of following students:** • **who pass with distinction (marks >= 75)** • **who just pass (marks >= 33 and marks < 75)** • **who fail (marks < 33)**
Steps	**1) Add "Pass", "Distinction" and "Fail" in HashMap and set their values to 0.** **2) Traverse the input array. For each number, based on its value, increment correct key's value.** **3) Print the HashMap.**

```
public void analyzeMarks(int[] marks)
{
      //print count of students who pass with distinction, just pass and fail
}
```

Test Cases	**Input**	**Output**
	`analyzeMarks({22,19,33,40,90,83,32,75})`	`{Distinction=3, Pass=2, Fail=3}`

Problem 5 CountDuplicates	Complete the function given below. The function takes an integer array as input. It then prints count of each number present in this array. It also prints unique numbers from the input array. Use HashMap to solve this problem.

```
public void countDuplicates(int[] arr)
{
	//prints count of each number in the input array
	//prints total unique numbers from the input array
}
```

Test Cases	Input	Output
	countDuplicates({1,1,1,2,3,2,5,3,2})	{1=3,2=3,3=1,5=1} Unique=4