HashMap is a class in the `java.utils` package that implements the **Map** interface. It is used to store the key-value pair. Let's suppose we need to store the stock prices of some companies. In this case, we can use a **HashMap**. The company name will be the key and the stock price will be the value.

Some of the features of HashMap are:

1. The keys should be unique.
2. HashMap allows only one `null` key.
3. The values can be null or duplicate.
4. The keys are stored in random order.

Creating a HashMap

There are four different constructors available to create a HashMap in Java.

Using the no-arg constructor

The simplest way to create a **HashMap** is by using the no-arg constructor. This constructor creates a **HashMap** with an initial capacity of 16 and **load factor** of 0.75

Below is the code syntax to create a **HashMap**. It states that the key is a String type and the *value* is an Integer type.

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

> Load factor is a number that defines when a Map should be resized. If the load factor is 0.75, then it means that the Map should be resized when it is 75 percent full.

Using the constructor that takes initial capacity

We can also provide the initial capacity while creating the **HashMap**. If we are already aware that our **HashMap** will contain more than 16 elements, then it is better to provide some initial capacity as it reduces the number of resizes. In this case, also, the default load factor (0.75) is used.

Using the constructor that takes initial capacity and load factor

We can also provide initial capacity with the load factor while creating the **HashMap**. If we don't want frequent resizing then we can set the load factor to a higher number.

Using the constructor that takes another Map as a parameter

We can also create a **HashMap** using another Map by passing it to the constructor. The newly created **HashMap** will have the same size as that of the passed Map, whereas the load factor will default to **0.75**

Inserting into a HashMap

Let's discuss all the methods that we can use to insert a key-value pair in a **HashMap**.

Using the `put()` method

We can use the `put(K key, V value)` method to insert a key-value pair in the **HashMap**. If the key is not present, then a new key-value pair will be added. If the key is already present, then the value will be updated.

Using the `putIfAbsent()` method

The `putIfAbsent(K key, V value)` method inserts a key-value pair only if it is not already present in the Map. If the key is already present then its value will not be updated. This method was added in Java 8.

Using the `putAll()` method

The `putAll(Map<? extends K, ? extends V> m)` method copies all of the mappings from the specified map to this map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map.

The below example shows **HashMap** working properly.

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

public class HashMapDemo {

    public static void main(String args[]) {

        Map<String, Integer> stockPrice = new HashMap<>();

        stockPrice.put("Oracle", 56); //Inserting share price of Oracle in the Map.
        System.out.println(stockPrice);

        stockPrice.put("Oracle", 60); //Inserting share price of Oracle again. This will update the value
        System.out.println(stockPrice);

        stockPrice.putIfAbsent("Oracle", 70); //Inserting share price of Oracle again using putIfAbsent()
        System.out.println(stockPrice);
    }
}
```

Run Save Reset