SYNTAX
TECHNOLOGIES

JAVA

Class 31

Agenda

Map Interface

Map

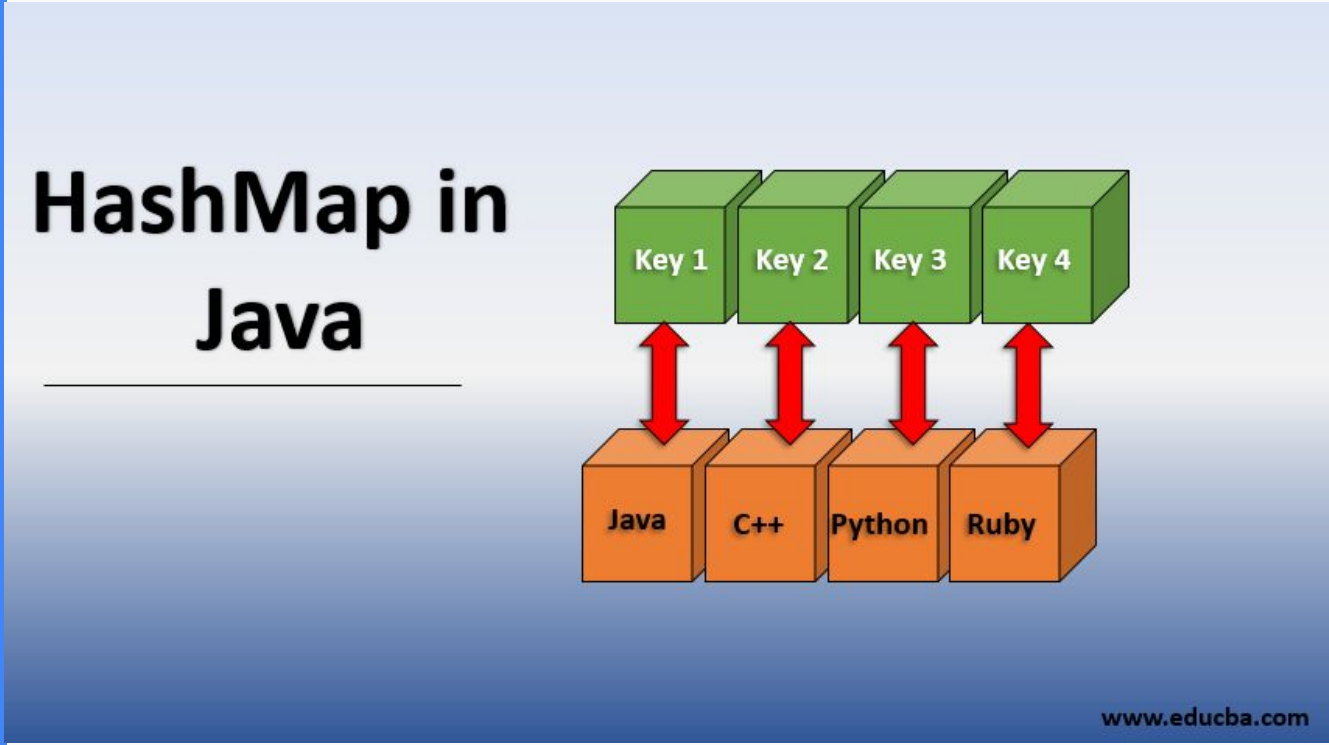
A Map is an object that maps keys to values.

Map stores the values in Key-Value pair and each pair is known as an entry.

Values in the Map can be duplicate but keys always must be unique

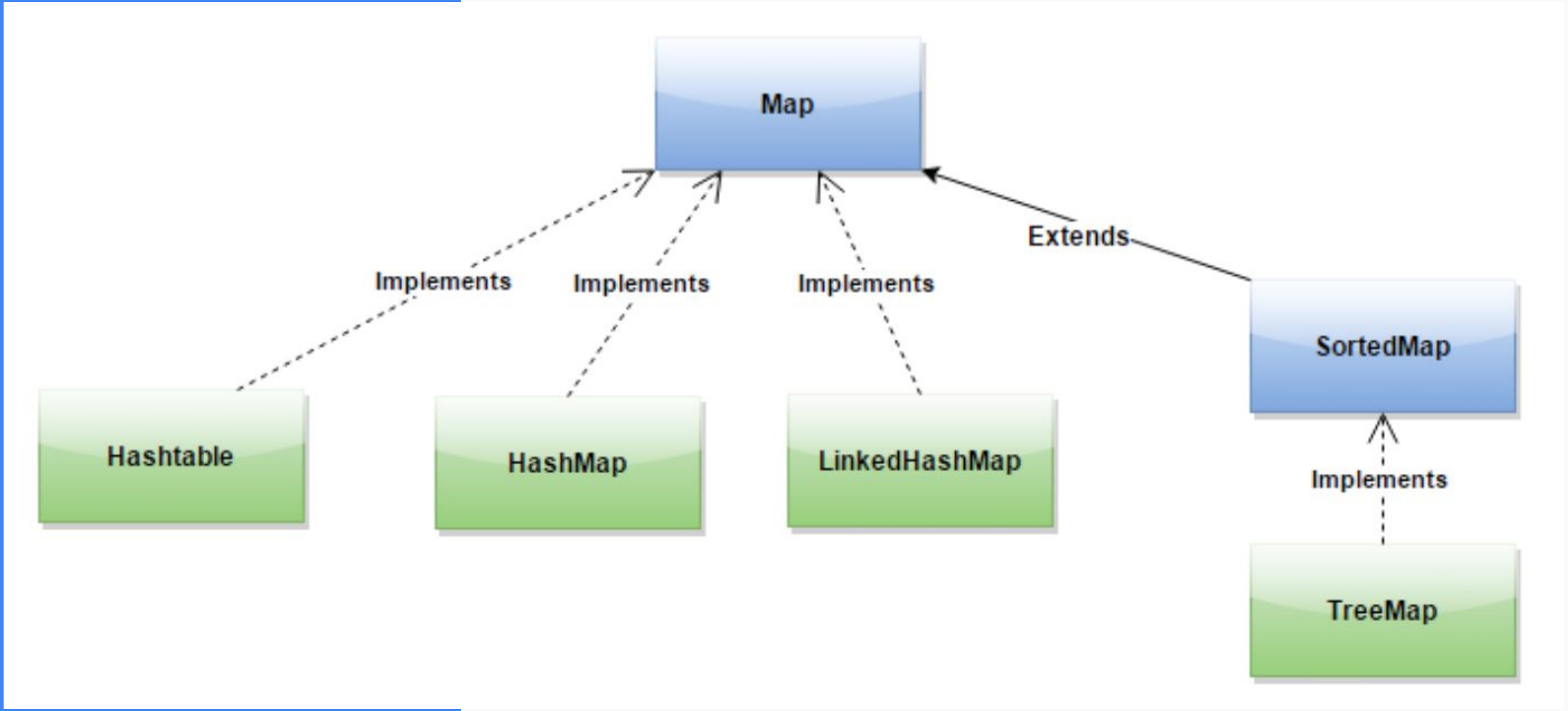
The Map interface is implemented by different Java classes, such as HashMap, LinkedHashMap, TreeMap and HashTable

How Maps are stored in the memory



Map

Maps are not part of collection but built based on the collection concepts



Map

A map has the form Map <K, V> where:

K : Specifies the Keys
V : Specifies the Values

Main implementations of Map interfaces.

HashMap: it makes no guarantees concerning the order of iteration, HashMap doesn’t maintain the insertion order of key elements.

LinkedHashMap: It orders key elements based on the order in which they were inserted into the set (insertion-order).

TreeMap: It orders its elements based on their key values; it is substantially slower than HashMap.

Methods in Map

- **public Object put(object key,Object value)**: This method is used to insert an entry in this map.
- **public void putAll(Map map)**: This method is used to insert the specified map in this map.
- **public Object remove(object key)**: This method is used to delete an entry for the specified key.
- **public Object get(Object key)**: This method is used to return the value for the specified key.
- **public boolean containsKey(Object key)**: This method is used to search the specified key from this map.
- **public boolean containsValue(Object value)**: This method is used to search the specified value from this map.

Map Methods

void clear(): It removes all the key and value pairs from the specified Map.
Object clone(): It returns a copy of all the mappings of a map and used for cloning them into another map.
boolean containsKey(Object key): It is a boolean function which returns true or false based on whether the specified key is found in the map.
boolean containsValue(Object Value): Similar to containsKey() method, however it looks for the specified value instead of key.
Value get(Object key): It returns the value for the specified key.
boolean isEmpty(): It checks whether the map is empty. If there are no key-value mapping present in the map then this function returns true else false.

HashMap in Java

- HashMap class implements the Map interface in Java.
- HashMap stores the elements in Key-Value pair.
- HashMap contains only unique elements i.e You can’t use the same key-value pair again.
- HashMap may have one null key and multiple null values.
- HashMap maintains no order.
- It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap.
- It neither does any kind of sorting to the stored keys and Values.
- You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

HashMap

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

studentMap.put(101, "John");
studentMap.put(102, "Jason");
studentMap.put(103, "Jordan");
studentMap.put(104, "Jenny");

//to check if specific key or value exist
System.out.println(studentMap.containsKey(101));
System.out.println(studentMap.containsValue("Jordan"));
//to access 1 value
System.out.println(studentMap.get(102));
//replace value
studentMap.replace(104, "Nikky");
System.out.println(studentMap);
//remove object
studentMap.remove(103);
System.out.println(studentMap);
```

HashMap in Java

```
public class CheckKeyExample {

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

    hashmap.put(11,"Chaitanya");
    hashmap.put(22,"Pratap");
    hashmap.put(44,"Rajesh");
    hashmap.put(55,"Kate");

    boolean flag = hashmap.containsKey(22);
    SOP("Key 22 exists in HashMap? : " + flag);

    boolean flag2 = hashmap.containsKey(55);
    SOP("Key 55 exists in HashMap? : " + flag2);

    boolean flag3 = hashmap.containsKey(99);
    SOP("Key 99 exists in HashMap? : " + flag3);
 }
}
```

```
public class CheckValueExample {

 public static void main(String[] args) {

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

    hashmap.put(11,"Chaitanya");
    hashmap.put(22,"Pratap");
    hashmap.put(33,"Singh");
    hashmap.put(44,"Rajesh");
    hashmap.put(55,"Kate");

boolean flag =
hashmap.containsValue("Singh");
    SOP(" String Singh exists in HashMap? : " +
flag);
 }
}
```

LinkedHashMap in Java

- LinkedHashMap implements Map Interface in Java. It stores the element in the LinkedList and maintains the Insertion order.
- That’s why, when iterating through a collection-view of a LinkedHashMap, the elements will be returned in the order in which they were inserted.
- Also, if we insert the key again into the LinkedHashMap the original orders are retained. This allows insertion-order iteration over the map.
- That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.

```
public class LinkedHashMapDemo {

public static void main(String args[]) {

 LinkedHashMap<Integer, String> lmap =
new LinkedHashMap<>();

// Add Elements in the HashMap
 lmap.put(1, "Anshul");
 lmap.put(12, "Mirilla");
 lmap.put(10, "Terrna");
 lmap.put(15, "Eden");
 lmap.put(19, "Matt");

 System.out.println("LinkedHashMap
before modification" + lmap);
```

```
System.out.println("Is Employee ID 15
exists: " + lmap.containsKey(15));
 System.out.println("Is Employee name
Terrna Exists: "
 + lmap.containsValue("Terrna"));
 System.out.println("Total number of
employees: " + lmap.size());
 System.out.println("Removing Employee
with ID 12: " + lmap.remove(12));
 System.out
 .println("Removing Employee with ID 33
(which does not exist): "
 + lmap.remove(33));
 System.out.println("LinkedHashMap After
modification" + lmap);
 }
}
```

TreeMap in Java

- TreeMap implements the Map interface and it is a collection that sorts its keys in the ascending order.
- It contains only unique elements.
- It cannot have null key but can have multiple null values.
- A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.
- TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations.

```
public class Details {

   public static void main(String args[]) {

      /* This is how to declare TreeMap */
      TreeMap<Integer, String> tmap =   new TreeMap<>();

      /*Adding elements to TreeMap*/
      tmap.put(1, "Data1");
      tmap.put(23, "Data2");
      tmap.put(70, "Data3");
      tmap.put(4, "Data4");
      tmap.put(2, "Data5");
 /* Display content using Iterator*/
      Set<Entry<Integer, String>>set = tmap.entrySet();
      Iterator<Entry<Integer, String>> iterator = set.iterator();
      while(iterator.hasNext()) {
         Entry<Integer, String>mentry = iterator.next();
         System.out.print("key is: "+ mentry.getKey() + " & Value is: ");
         System.out.println(mentry.getValue());
      }
   }
}
```

HashTable in Java

- Hashtable class implements a hashtable, which maps keys to values.
- HashTable implements the Map interface in Java.
- Like other Map classes, it also stores the data in a key-value pair but the HashTable store only non-null objects i.e any key or value can’t be Null.
- Hashtable is similar to HashMap except it is synchronized or we say thread safe.

HashTable in Java

```
public class HashTableDemo {
	public static void main(String[] args) {
		// Creating a Hashtable
		Hashtable<String, String> hashtable = new Hashtable<>();

		// Adding Key and Value pairs to HashTable
		hashtable.put("Key1", "Chaitanya");
		hashtable.put("Key2", "Ajeet");
		hashtable.put("Key3", "Peter");
		hashtable.put("Key4", "Ricky");
		hashtable.put("Key5", "Mona");

		System.out.println(hashtable);

		// Retrieving all keys and values from HashTable
		for (String key : hashtable.keySet()) {
			System.out.println(key + ":" + hashtable.get(key));
		}
	}
}
```

Map Methods

Set .keySet(): It returns the Set of the keys fetched from the map.

value .put(Key k, Value v): Inserts key value mapping into the map. Used in the above example.

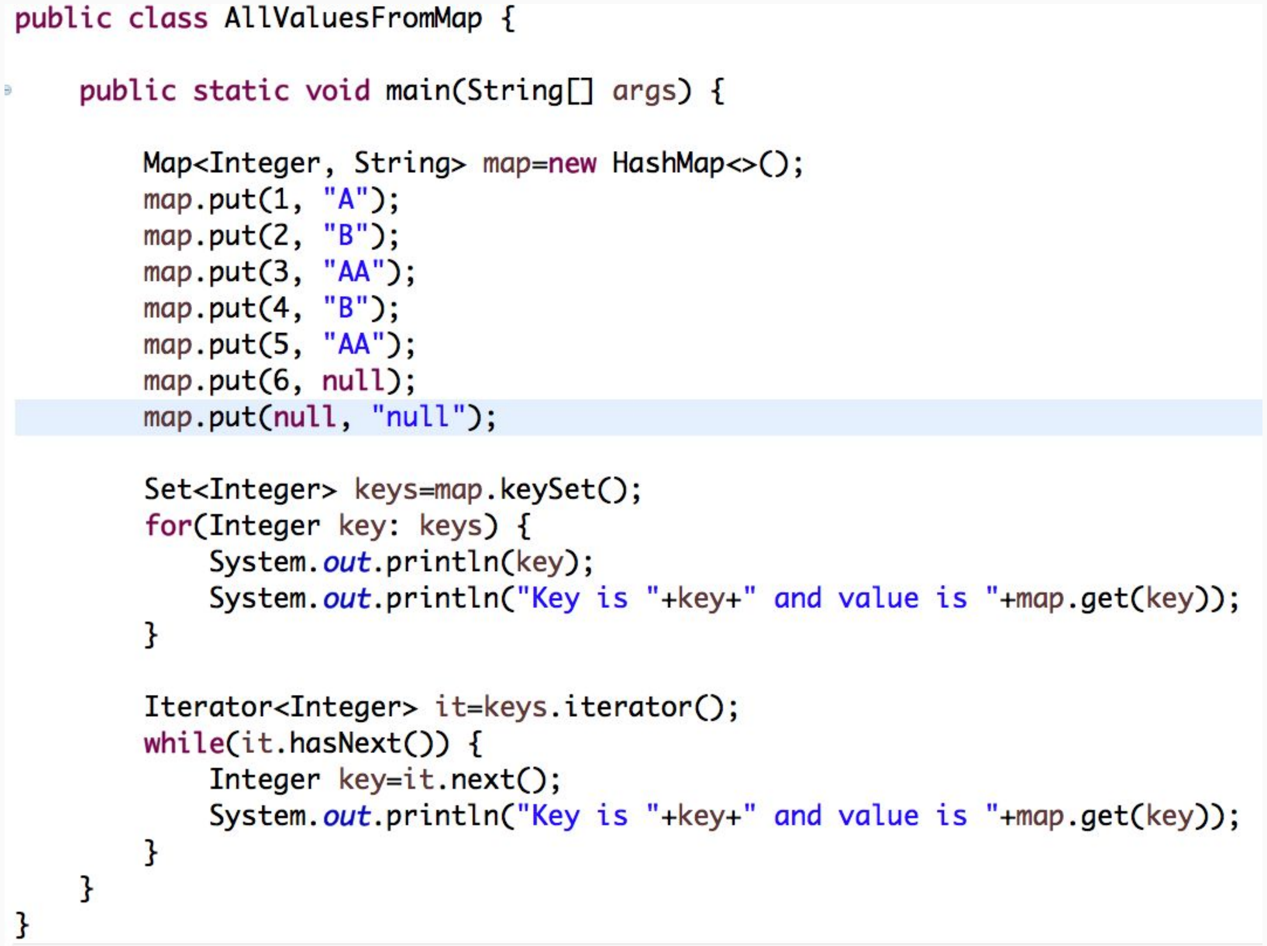
int .size(): Returns the size of the map – Number of key-value mappings.

Collection .values(): It returns a collection of values of map.

Value .remove(Object key): It removes the key-value pair for the specified key. Used in the above example.

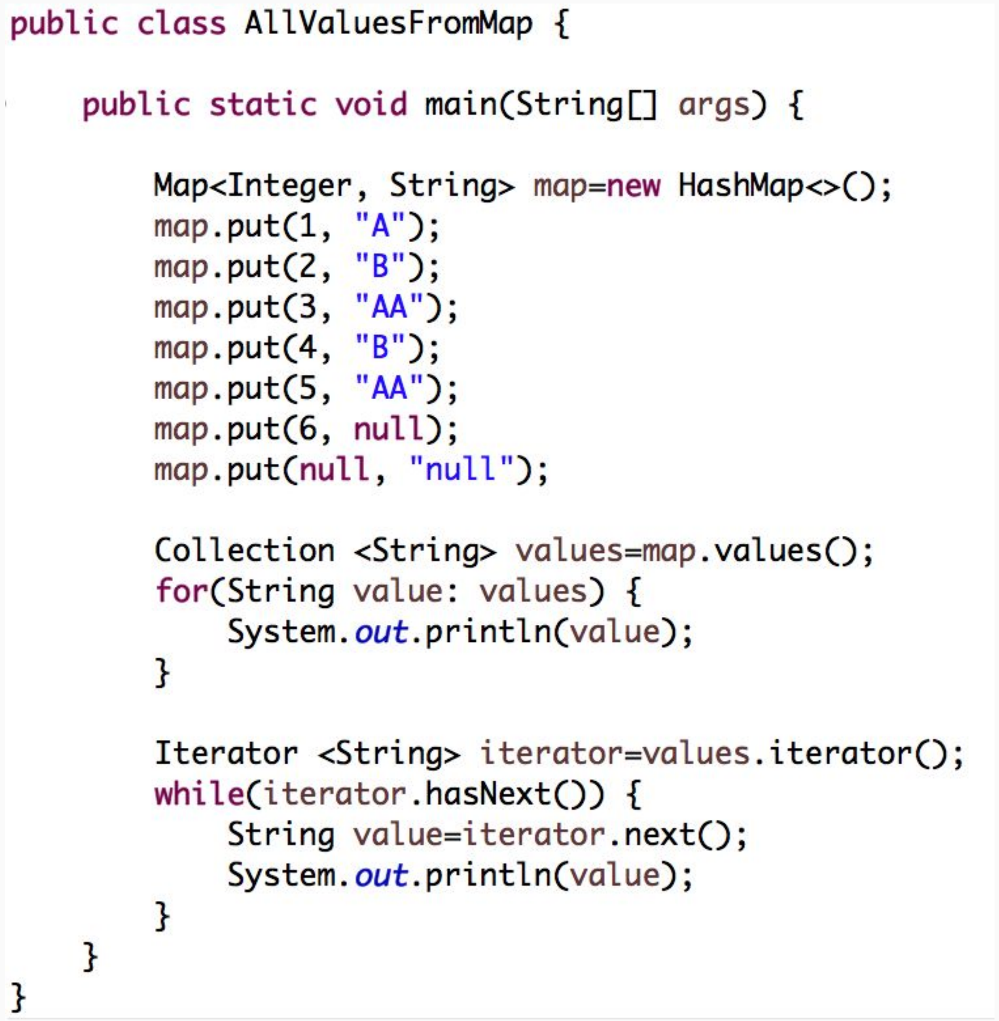
void .putAll(Map m): Copies all the elements of a map to the another specified map.

How to get all Keys from a Map



```
public class AllValuesFromMap {

    public static void main(String[] args) {

        Map<Integer, String> map=new HashMap<>();
        map.put(1, "A");
        map.put(2, "B");
        map.put(3, "AA");
        map.put(4, "B");
        map.put(5, "AA");
        map.put(6, null);
        map.put(null, "null");

        Set<Integer> keys=map.keySet();
        for(Integer key: keys) {
            System.out.println(key);
            System.out.println("Key is "+key+" and value is "+map.get(key));
        }

        Iterator<Integer> it=keys.iterator();
        while(it.hasNext()) {
            Integer key=it.next();
            System.out.println("Key is "+key+" and value is "+map.get(key));
        }
    }
}
```

How to get all Values from a Map



```java
public class AllValuesFromMap {

    public static void main(String[] args) {

        Map<Integer, String> map=new HashMap<>();
        map.put(1, "A");
        map.put(2, "B");
        map.put(3, "AA");
        map.put(4, "B");
        map.put(5, "AA");
        map.put(6, null);
        map.put(null, "null");

        Collection <String> values=map.values();
        for(String value: values) {
            System.out.println(value);
        }

        Iterator <String> iterator=values.iterator();
        while(iterator.hasNext()) {
            String value=iterator.next();
            System.out.println(value);
        }
    }
}
```

How to loop Map in java

```
Map<String, Integer> classroomMap=new LinkedHashMap<>();
classroomMap.put("Table", 20);//Entry<Key, Value>
classroomMap.put("Chair", 60);//Entry<String, Integer>
classroomMap.put("Screen", 3);
classroomMap.put("Student", 60);
classroomMap.put("Instructor", 3);

Set<Entry<String, Integer>> entrySet=classRoom.entrySet();
System.out.println("---------using for each loop to get all entry objects");

    for(Entry<String, Integer> entry:entrySet) {
              System.out.println(entry.getKey()+":"+entry.getValue());
    }
System.out.println("---------using iterator to get all entry objects");
Iterator<Entry<String, Integer>> entryIterator=entrySet.iterator();
         while(entryIterator.hasNext()) {
                  Entry<String, Integer> ent=entryIterator.next();
                  String entry=ent.getKey()+"----"+ent.getValue();
                  System.out.println(entry);
          }
```