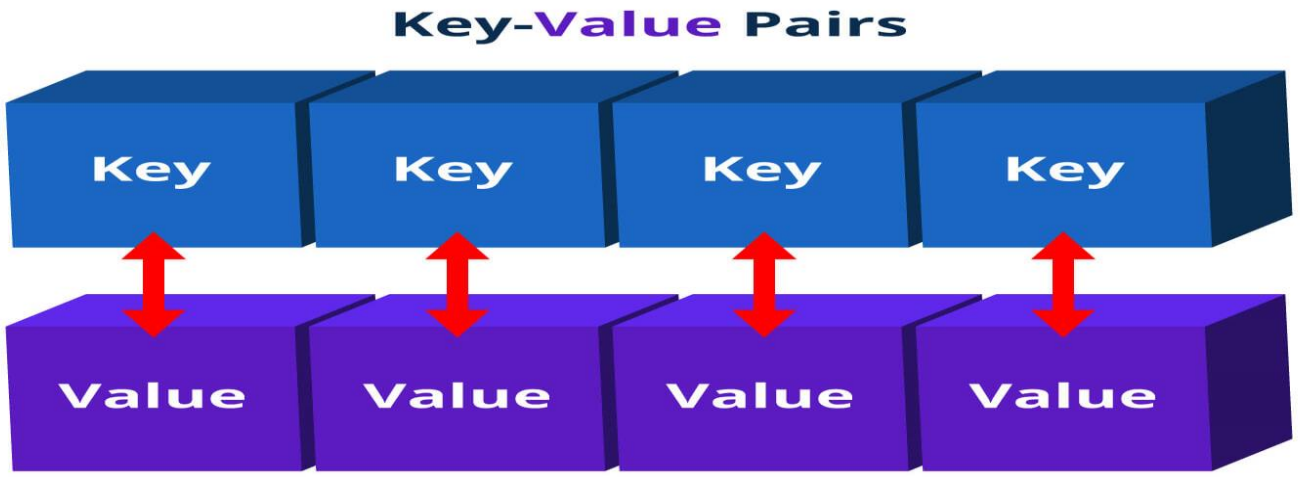
COLLECTIONS

(An easy way to manage Objects)

The Map Interface

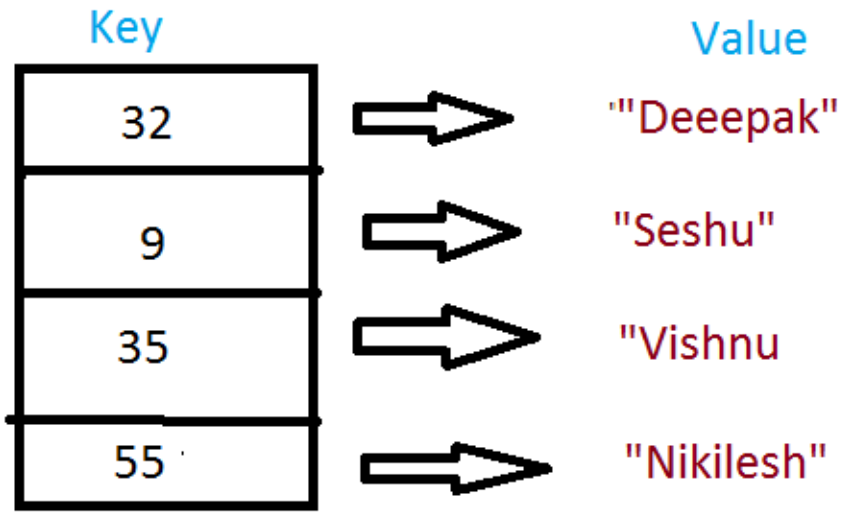
- It is **not** the **child interface** of **Collection** interface but has **methods** similar to **Collection**

- A **Map** is an **object** that **stores data in pairs** , called **key-value** pair.

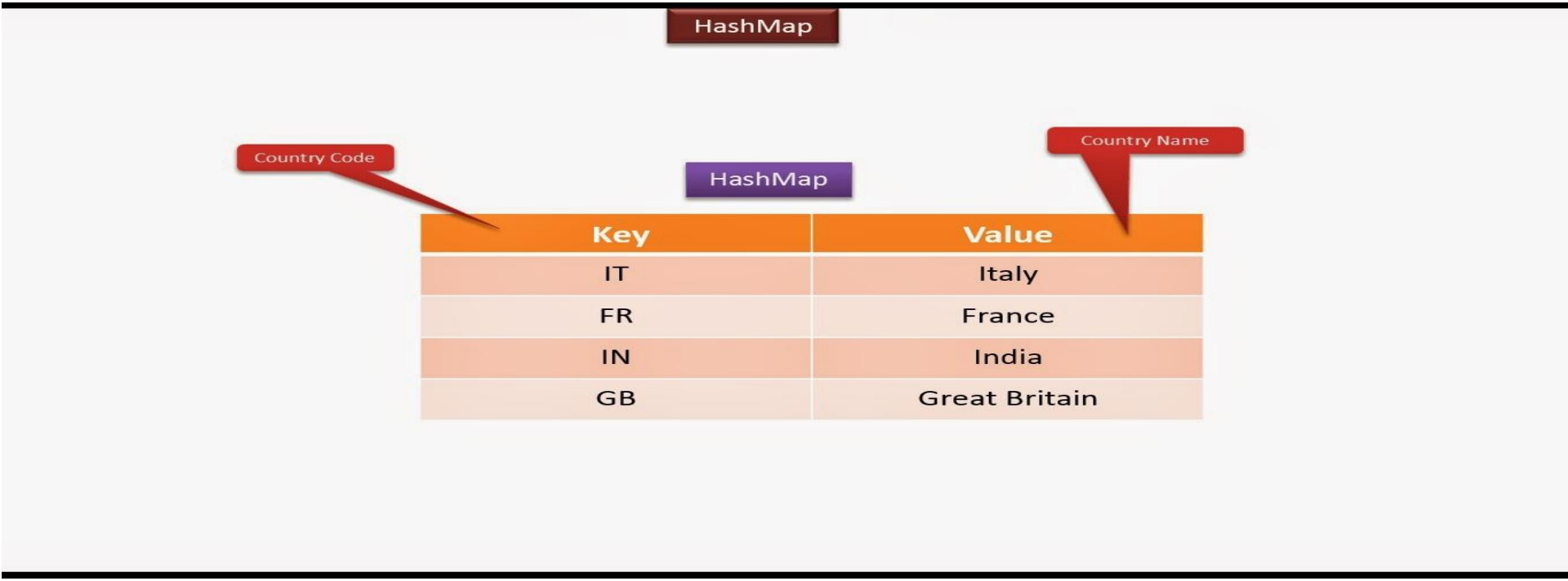


Example Of Key-value Pair

Example Of Key-value Pair



Example Of Key-value Pair



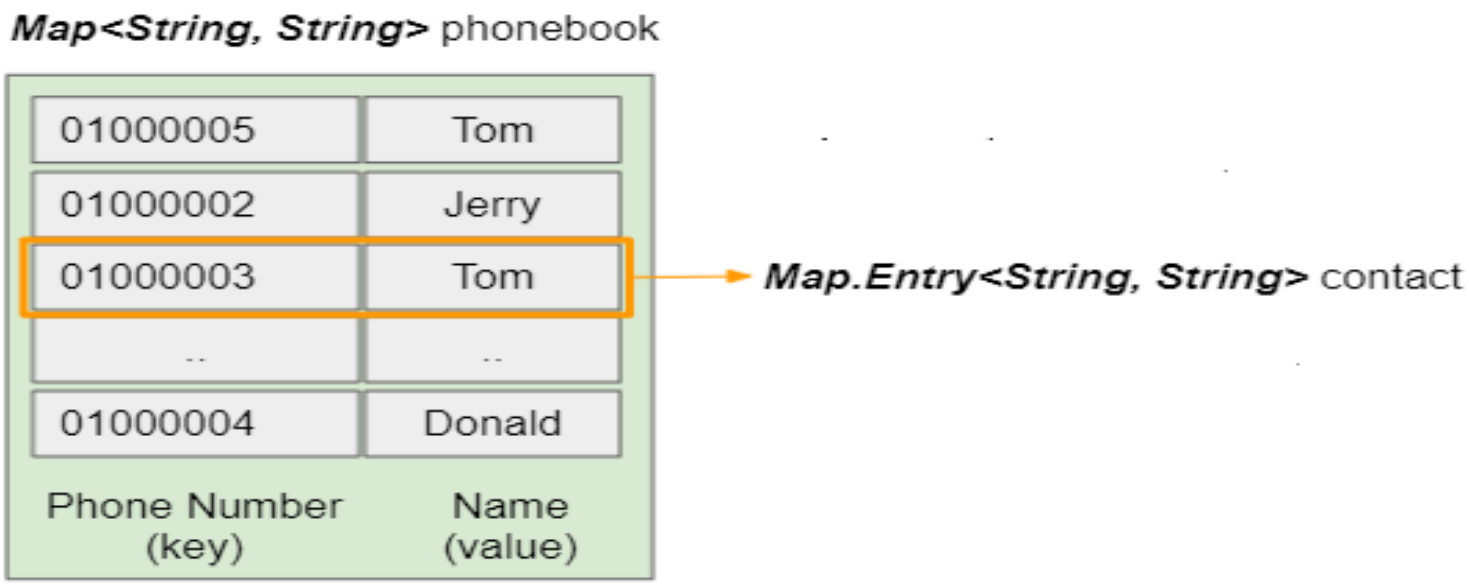
How values are stored in Map ?

1.The **keys** in a **Map** have to be **unique**.

2.Each **key-value** pair is **saved** in the **Map** is **saved** as an object of type **Entry**

How values are stored in Map ?

How values are stored in Map ?



Important Methods Of Map

- **Object put(Object k, Object v)**

1. Puts an entry in the invoking map, overwriting any previous value associated with the key.

2. The key and value are **k** and **v**, respectively.

3. Returns **null** if the key did not already exist , otherwise it **returns the previous value** linked to the key

Important Methods Of Map

- **Object get(Object k)**

1. Returns the value associated with the key **k**
2. If the key is not found , it returns **null**.

Important Methods Of Map

- **void clear( )**

1. Removes all key/value pairs from the invoking map.

Important Methods Of Map

- **boolean containsKey(Object k)**

1. Returns true if the invoking map contains **k** as a key.
2. Otherwise, returns false.

- **boolean containsValue(Object v)**

1. Returns true if the map contains **v** as a value.
2. Otherwise, returns false

Important Methods Of Map

- **boolean isEmpty( )**

1. Returns **true** if the invoking map is empty.
2. Otherwise, returns **false**.

- **int size( )**

1. Returns the number of key/value pairs in the map.

Important Methods Of Map

- **Object remove(Object k)**

1. Removes the entry whose key equals **k**
2. Returns the previous value associated with the specified key,
3. Otherwise it returns **null** if there was no mapping for the key

Important Methods Of Map

- **Collection values( )**

1. Returns a Collection containing the values in the map.

Important Methods Of Map

- **Set keySet( )**

1. Returns a Set that contains the keys in the invoking map.

Important Methods Of Map

- **Set entrySet( )**

1. Returns a **Set** that contains the entries in the map.

2. The set contains objects of type **Map.Entry**.

Important Methods Of Entry

- **Object getKey( )**

1. Returns the **key** corresponding to this **Entry** object

Important Methods Of Entry

- **Object getValue( )**

1. Returns the **value** corresponding to this **Entry** object

Important Methods Of Entry

- **Object setValue(Object)**

1. Replaces the **old value** corresponding to this entry with the specified **value**
2. Returns the **old value**

Implementation Classes

The Collections Framework provides 2 very important Map implementation:

1 - HashMap

2 - TreeMap

HashMap:

The HashMap is a class which is used to perform some basic operations such as inserting, deleting, and locating elements in a Map

TreeMap:

The TreeMap implementation is useful when we need to traverse the keys from a collection in a sorted manner. The elements added to a TreeMap must be **sortable** in order to work properly.

The HashMap class

- Internally uses **HashTable** to store the data.
- Stores data as **key-value** pairs
- It contains only **unique** elements.
- It is not an ordered collection.

The HashMap class

- It neither does any kind of sorting to the stored keys and Values.

- Ensures retrieval of the object on constant time i.e. O(1)

The HashMap Constructors

- **HashMap( )**

  This constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).

- **HashMap(int initialCapacity)**

Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).

The HashMap Constructors

- **HashMap(int initalCapacity,float loadFactor)**

  Constructs an empty HashMap with the specified initial capacity and load factor.

- **HashMap(Map m)**

Constructs a new HashMap with the same mappings as the specified Map.

Exercise 9

WAP to store the **Names** and **Phone Numbers** of following members of **TEAM SCA** . Now retrieve these values and display them

Name	Phone
Sachin	**9826086245**
Aftaab	**7992202926**
Arif	**8982585147**
Mohnish	**8962336876**

Adding Data In HashMap

```
Map<String,Long> teamSca = new HashMap<>();

teamSca.put("Sachin", 9826086245L);

teamSca.put("Aftaab",7992202926L );

teamSca.put("Arif", 8982585147L);

teamSca.put("Mohnish", 8962336876L);
```

Retrieving Data From HashMap

- Retrieval of data from **HashSet** can be done in **4 ways**:
  - **Retrieving all values together**
  - **Retrieving only keys**
  - **Retrieving only values**
  - **Retrieving key-value pairs**

Retrieving All Values Together

For this we simply have to pass the name of **HashSet** reference to the method **println()**

Example:

```
Map<String,Long> teamSca = new HashMap<>();
teamSca.put("Sachin", 9826086245L);
teamSca.put("Aftaab",7992202926L );
teamSca.put("Arif", 8982585147L);
teamSca.put("Mohnish", 8962336876L);
System.out.println(teamSca);
```

Output:

```
{Mohnish=8962336876, Aftaab=7992202926, Sachin=9826086245,
  Arif=8982585147}
```

Retrieving Only The Keys

For this we have to do 2 things:

1. **Call the method keySet() which returns a Set of keys of the HashMap**
2. **Use an Iterator over this Set**

Example:

```
Map<String,Long> teamSca = new HashMap<>();
.......
Set s=teamSca.keySet();
Iterator <String> it=s.iterator();
while(it.hasNext())
{
   String key=it.next();
   System.out.println(key);
}
```

Output:

```
Mohnish
Aftaab
Sachin
Arif
```

Retrieving Only The Values

For this we have to do 2 things:

1. **Call the method values() which returns a Collection of values of the HashMap**
2. **Use an Iterator over this Collection**

Example:

```
Map<String,Long> teamSca = new HashMap<>();
.......
Collection c=teamSca.values();
Iterator <Long> it=c.iterator();
while(it.hasNext())
{
   Long value=it.next();
   System.out.println(value);
}
```

Output:

```
8962336876
7992202926
9826086245
8982585147
```

Retrieving Key-Value Pairs

For this we have to do 2 things:

1. Call the method **entrySet( )** which returns a **Set** of all the data in the **HashMap**
2. Each element in this Set is an object of type **Entry**
3. **Entry** is an inner interface of **Map** and has ***2 methods*** called **getKey()** and **getValue()**
4. So we will have to get an **Iterator** over this **Entry**

Retrieving Key-Value Pairs

Example:

```
Map<String,Long> teamSca = new HashMap<>();
.......
Set e=teamSca.entrySet();
Iterator  it=e.iterator();
while(it.hasNext())
{
    Entry et=(Map.Entry)it.next();
    System.out.println(et.getKey()+","+et.getValue());
}
```

Output:

```
Mohnish,8962336876,
Aftaab,7992202926,
Sachin,9826086245
Arif,8982585147
```

Checking whether a value exists or not

To get the Value from **HashMap** object we use the method:

boolean containsValue(value)

This method returns **true** if list contains the specified Value otherwise returns **false**.

Program

```
import java.util.*;
public class HashMapDemo{
        public static void main(String args[]) {
                Map<String,Long> teamSca = new HashMap<>();
                teamSca.put("Sachin", 9826086245L);
                teamSca.put("Aftaab",7992202926L );
                teamSca.put("Arif", 8982585147L);
                teamSca.put("Mohnish", 8962336876L);

                boolean bool = teamSca.containsKey(“Sachin");
                System.out.println(“Does Sachin is exists in HashMap : "
+ bool);

        }
}
```

Output:

Does Sachin is exists in HashMap : true

Using remove( ) and size( )

To get total number of elements in a **HashMap** we use the method:

public int size()

To remove a particular key from the **HashMap** we use the method:

public Object remove(key)

This method removes the specified entry from the **HashMap** whose key is passed as argument and returns the deleted value

```
import java.util.*;
public class HashMapDemo {
	public static void main(String args[]) {
		Map<String,Long> teamSca = new HashMap<>();
		teamSca.put("Sachin", 9826086245L);
		teamSca.put("Aftaab",7992202926L );
		teamSca.put("Arif", 8982585147L);
		teamSca.put("Mohnish", 8962336876L);
		System.out.println("The size of HashMap is : " + hm.size());
		hm.remove(“Arif");
		System.out.println("The size of HashMap after alteration is :
"
				+ hm.size());

	}
}
```

Output

Output:

The size of HashMap is : 4

The size of HashMap after alteration is : 3

The TreeMap class

- **TreeMap** class implements **NavigableMap** , **SortedMap** and **Map** interfaces.

The TreeMap class

1.The **TreeMap** class implements the **Map** interface by using a tree.

2. A **TreeMap** provides an efficient means of storing key/value pairs in **sorted order**

3. Doesn’t store duplicates

Example

```
import java.util.*;
class TreeMapDemo {
public static void main(String args[]) {
Map<String,Long> teamSca = new TreeMap<>();
teamSca.put("Sachin", 9826086245L);
teamSca.put("Aftaab",7992202926L );
teamSca.put("Arif", 8982585147L);
teamSca.put("Mohnish", 8962336876L);
System.out.println(teamSca);
}
}
```

Output

{Aftaab=7992202926, Arif=8982585147, Mohnish=8962336876, Sachin=9826086245}

Example

```
import java.util.*;
class TreeMapDemo {
public static void main(String args[]) {
Map<String,Long> teamSca = new TreeMap<>();
teamSca.put("Sachin", 9826086245L);
teamSca.put("Aftaab",7992202926L );
teamSca.put("Arif", 8982585147L);
teamSca.put("Mohnish", 8962336876L);
```

```
Set e=teamSca.entrySet();
Iterator  it=e.iterator();
while(it.hasNext())
{
    Entry et=(Map.Entry)it.next();
    System.out.println(et.getKey()+","+et.getValue());
}
}
}
```

Output

Aftaab,7992202926,

Arif,8982585147

Mohnish,8962336876

Sachin,9826086245

Exercise 10

A Bank maintains bank account of it's users in the database. When you visit the bank and tell your account number to the teller , he fetches your account details.

1. Create a collection and maintain bank accounts in that.
2. **There should be a single entry for one account number**
3. **We should be able to fetch the account details when an account number is supplied to the collection.**

The program should have 3 classes:

1. **Account**: should contain 3 data members called **accountNumber**, **name** and **balance**. Also provide appropriate constructor and **other important methods**

2. **Bank**: should contain a **HashMap** of **Account**. Also provide 5 methods called :
    1. **addAccount()**
    2. **getAccount()**
    3. **removeAccount()**
    4. **getCount()**
    5. **getAllAccounts()**

3. **UseBank:** This will be our **driver class**. It will contain code to do the following:
    1. Create 4 **Account** objects and add them to the **Bank**.
    2. Display their details.
    3. Now fetch the details of a particular account by passing it's account number
    4. Remove an account by passing it's account number
    5. Display total number of Accounts

Exercise 11

Make changes in the Bank Application so that whenever we display the records , they always get displayed in sorted order of account id

HashMap v/s TreeMap

1. HashMap is useful when we need to access the map without considering how they are added to the map (means, unordered lookup of values using their keys).

2. HashMap doesn't allow duplicated entries.

3. TreeMap stores the elements in a tree.

4. TreeMap allows us to retrieve the elements in some sorted order .

5. So we can say that TreeMap is slower than HashMap