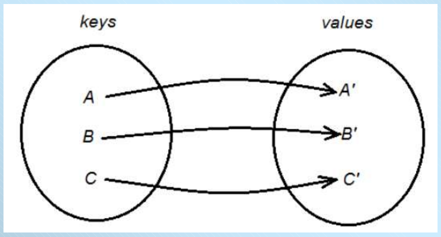
Collections

Map

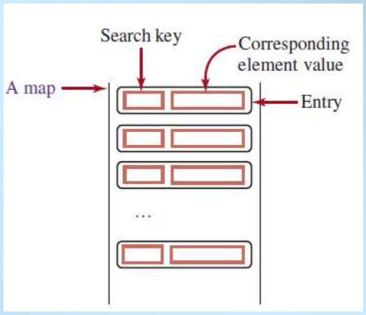


Objectives

- At the end of this topic, you should be able to
  - Understand the concept of Map.
  - Explain Map and its common operations.
  - Use the concrete implementations of Map in the Java collection framework .

What is a Map?

- A Map is a collection of entries or key/value pairs.
- Each entry is actually TWO objects: a key & a value.
- Each key maps to one value (like an index in List but not necessarily an Integer).
- Cannot contain duplicate keys.



Example uses of Map

Maps are suitable for key-value association mapping such as:

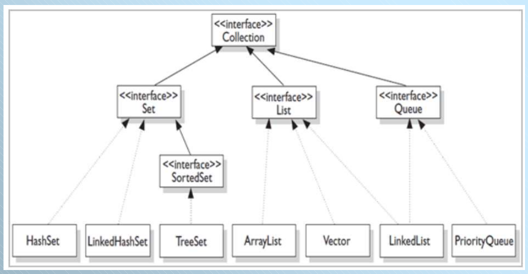
- Dictionaries.
- A map of zip codes and cities.
- A map of ic numbers and person data.
- Store word frequencies in a text

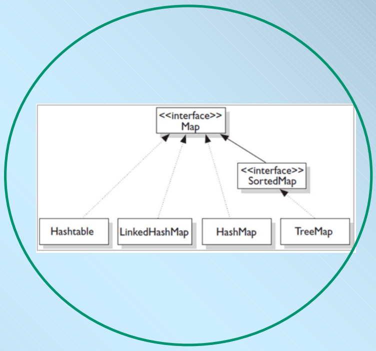
Map in Java

- Map is part of the Java Collection Framework BUT does not extend the Collection interface
- The Map interface provides basic methods for updating, querying, and obtaining a collection of values and a set of keys.
- Map implementations in Java:
  - HashMap, LinkedHashMap and TreeMap.

Map in Java

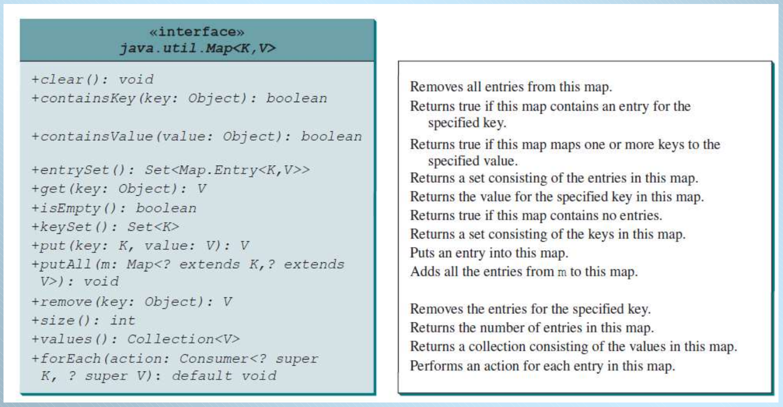
- Map is part of the Java Collection Framework BUT does not extend the Collection interface





Map in Java

- The Map interface:

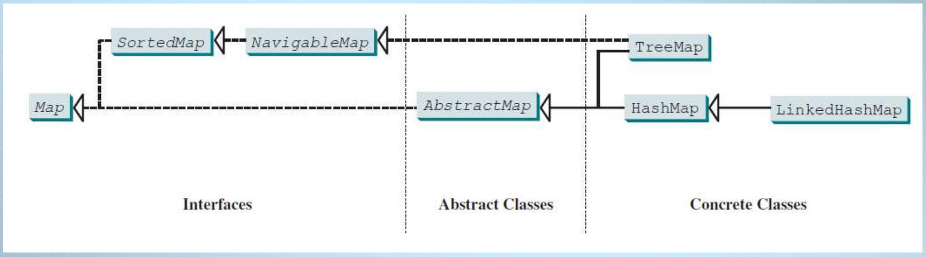


«interface»
java.util.Map<K,V>

`+clear(): void`	Removes all entries from this map.
`+containsKey(key: Object): boolean`	Returns true if this map contains an entry for the specified key.
`+containsValue(value: Object): boolean`	Returns true if this map maps one or more keys to the specified value.
`+entrySet(): Set<Map.Entry<K,V>>`	Returns a set consisting of the entries in this map.
`+get(key: Object): V`	Returns the value for the specified key in this map.
`+isEmpty(): boolean`	Returns true if this map contains no entries.
`+keySet(): Set<K>`	Returns a set consisting of the keys in this map.
`+put(key: K, value: V): V`	Puts an entry into this map.
`+putAll(m: Map<? extends K,? extends V>): void`	Adds all the entries from m to this map.
`+remove(key: Object): V`	Removes the entries for the specified key.
`+size(): int`	Returns the number of entries in this map.
`+values(): Collection<V>`	Returns a collection consisting of the values in this map.
`+forEach(action: Consumer<? super K, ? super V>): default void`	Performs an action for each entry in this map.

Map in Java

- 3 concrete Map classes:



HashMap

- Implemented using a hash table.
- Does not guarantee the order of its key-value elements
- Efficient for locating a value, inserting an entry, and deleting an entry

Creating and Initializing a Map

- A Map can be created using the "new" keyword and one of the Map implementation classes constructor.
- Need to specify two type parameters:

  <key type, value type>
- A Map can also be initialized using another Map object.

```
Map<String,String> dictionary = new HashMap<>();  // create a HashMap of
                                                   //String keys & String values
Map<Integer, Person> emp = new HashMap<>();   // create a HashMap of Integer keys
                                               //Person values

Map<String,String> dictionary2 = new HashMap<>(dictionary);  // create a HashMap
                                                              //from an existing map
```

Adding Entries to a Map

- An entry can be added to a Map using the “put" method that takes 2 arguments: (key, value).
- If the key already exist, the old value for the key is replaced by the specified value

```
Map<String,String> phones = new HashMap<>();
phones.put(“John“,”012-8736868”);
phones.put(“Ali“,”019-4356226”);
phones.put(“Lim“,”017-3767171”);
phones.put("Paul“,”010-3456789”);

System.out.println(phones);
```

Output:

```
{Lim=017-3767171, John=012-8736868, Paul=010-3456789, Ali=019-4356226}
```

Adding Entries to a HashMap

- When an entry is added to a HashMap, the key hash code is computed using the hashCode() method
- If the key hashcode do not match with existing keys, the entry is added to the hashtable.
- If there is a match, the equals() method is used to determine if the keys are equal.
- If equals() returns false, the entry is added.
- If equals() returns true, the key is considered a duplicate and its value replaces the existing value in the table.

Adding Entries to a HashMap

- If the key is an object from a user-defined class, the hashCode() and equals() MUST be overridden
- Refer to similar sections for adding user defined elements in Set

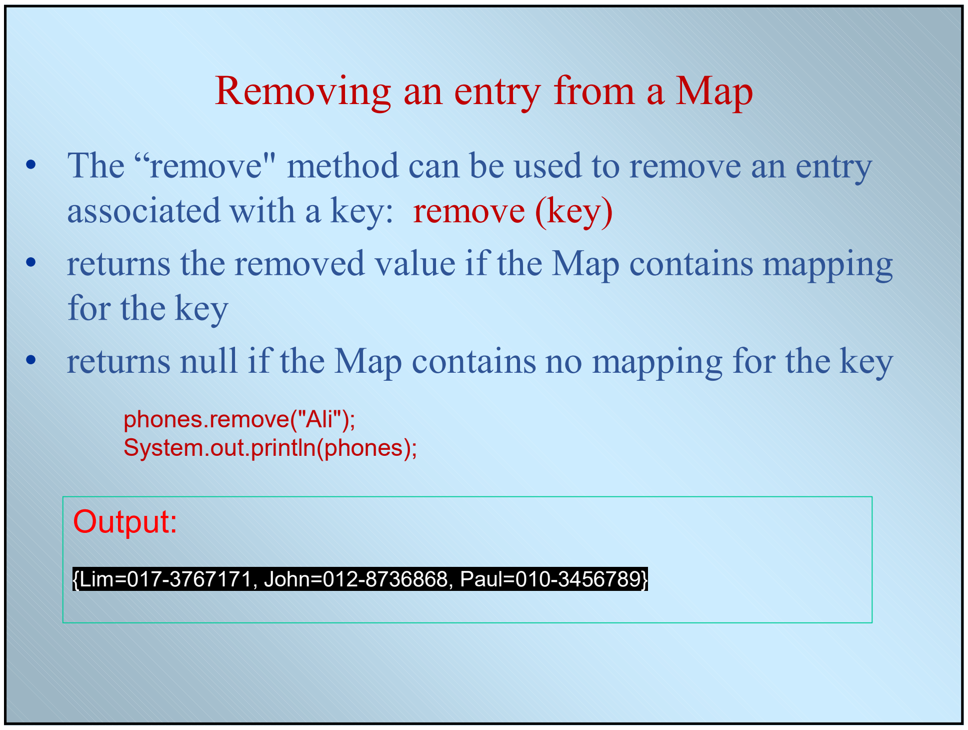
Getting a value in a Map

- The “get" method can be used to get a value associated with a key: get (key)
- returns null if the Map contains no mapping for the key

```
System.out.println("Ali contact no: "+phones.get("Ali"));
System.out.println("Abu contact no: "+phones.get("Abu"));
```

Output:

```
Ali contact no: 019-4356226
Abu contact no: null
```



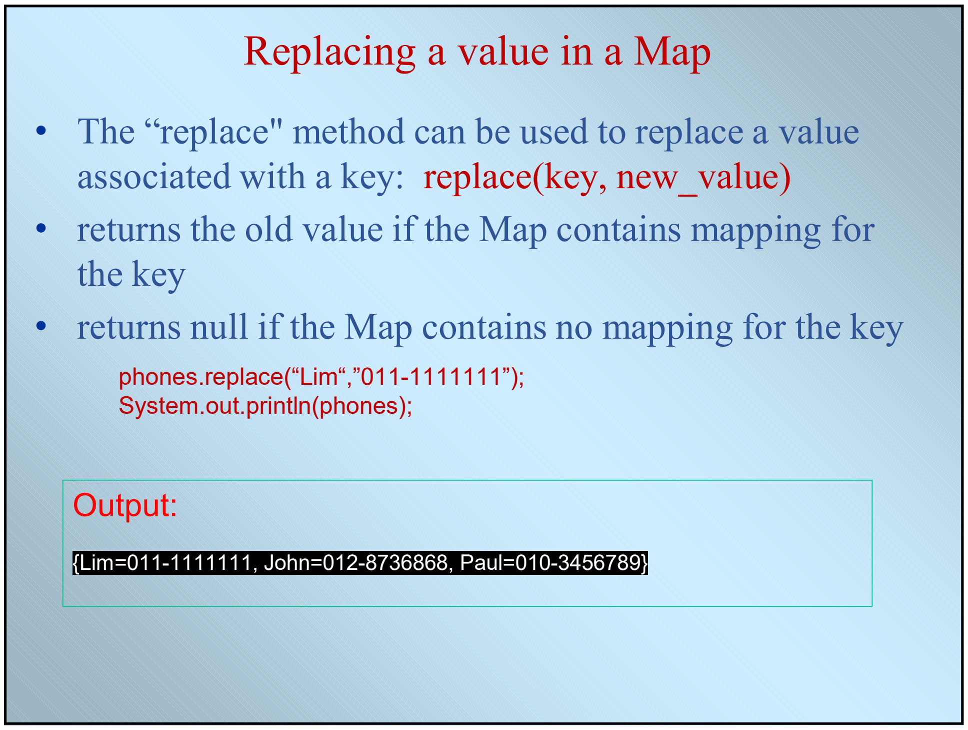
Removing an entry from a Map

- The “remove" method can be used to remove an entry associated with a key: remove (key)
- returns the removed value if the Map contains mapping for the key
- returns null if the Map contains no mapping for the key

```
phones.remove("Ali");
System.out.println(phones);
```

Output:

```
{Lim=017-3767171, John=012-8736868, Paul=010-3456789}
```



Replacing a value in a Map

- The “replace" method can be used to replace a value associated with a key: replace(key, new_value)
- returns the old value if the Map contains mapping for the key
- returns null if the Map contains no mapping for the key

```
phones.replace(“Lim“,”011-1111111”);
System.out.println(phones);
```

Output:

```
{Lim=011-1111111, John=012-8736868, Paul=010-3456789}
```

Querying a Map

- The "containsKey" method can be used to check if a Map contains a specified key: containsKey(key)
- The "containsValue" method can be used to check if a Map contains a specified value: containsValue(value)
- The "size" method can be used to get the number of entries in a Map: size()
- The "isEmpty" method can be used to check if a Map is empty: isEmpty()

Iterating a Map

- Map interface does not extend the Collection interface. Thus, a Map cannot be iterated directly using iterators or for-each loop.
- A Map can be viewed as a Collection by using these methods:
  - **keyset():** returns a Set of keys contained in the Map.
  - **values():** returns a Collection of values contained in the Map.
  - **entrySet():** returns a Set of entries (as key-value pairs) in the Map.

Iterating a Map

- The "keyset" method is used for iteration over the keys contained in the map. It returns the Set view of the keys: Set<K> keySet()

```
 Set<String> names = phones.keySet();
for (String person : names) {
         System.out.println("Name: "+person);
}
```

Output:

```
Name: Lim
Name: John
Name: Paul
```

Iterating a Map

- The "values" method is used for iteration over the values contained in the map. It returns the Collection view of the values: Collection<V> values()

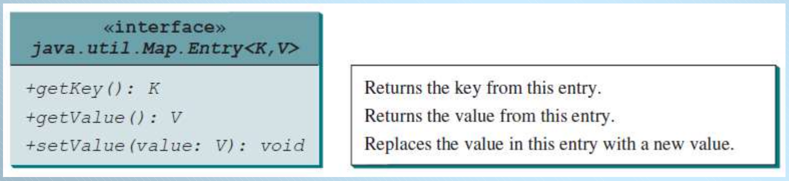
```
Collection<String> telNumbers = phones.values();
 for (String number : telNumbers) {
          System.out.println("Tel: "+number);
  }
```

Output:

```
Tel: 011-1111111
Tel: 012-8736868
Tel: 010-3456789
```

Iterating a Map

- The "entrySet" method is used for iteration over the entries (key-value pairs) contained in the Map. It returns the Set view of the Entry objects:
  Set<Map.Entry<K,V>>   entrySet()
- Entry is an (inner) interface defined in the Map interface


«interface»
java.util.Map.Entry<K,V>
+getKey(): K
+getValue(): V
+setValue(value: V): void
Returns the key from this entry.
Returns the value from this entry.
Replaces the value in this entry with a new value.

Iterating a Map

The Set of entries returned by entrySet() method can be iterated using a for-each loop:

```
Set<Map.Entry<String, String>> telEntries = phones.entrySet();
for (Map.Entry<String,String> entry  :  telEntries) {
        System.out.println("Name:"+entry.getKey()+", Tel: "+entry.getValue());
 }
```

Output:

```
Name:Lim, Tel: 011-1111111
Name:John, Tel: 012-8736868
Name:Paul, Tel: 010-3456789
```

LinkedHashMap in Java Map

- Implemented using a hash table and a linked list.
- Maintains the insertion order of the entries

```
Map<String,String> phones = new LinkedHashMap<>();
phones.put("John","012-8736868");
phones.put("Ali","019-4356226");
phones.put("Lim","017-3767171");
phones.put("Paul","010-3456789");

System.out.println(phones);
```

Output:

```
{John=012-8736868, Ali=019-4356226, Lim=017-3767171, Paul=010-3456789}
```

TreeMap in Java Map

- Implemented using a tree data structure.
- Automatically sorts according to the natural ordering of the keys (alphabetical or numerical order)
- Custom order of keys must be specified by:
  - implementing the Comparable interface for keys

    OR
  - passing an object that implements the Comparator object to the TreeMap constructor

TreeMap in Java Map

```
Map<String,String> phones = new TreeMap<>();
phones.put(“John“,”012-8736868”);
phones.put(“Ali“,”019-4356226”);
phones.put(“Lim“,”017-3767171”);
phones.put("Paul“,”010-3456789”);

System.out.println(phones);
```

Output:

```
{Ali=019-4356226, John=012-8736868, Lim=017-3767171, Paul=010-3456789}
```

Conclusion

- Java Map is an interface in the Java Collection Framework that stores key-value pairs and provides an efficient way to store, access, and manipulate data based on keys.
- If you don’t need to maintain an order in a map when updating it, use a HashMap.
- When you need to maintain the insertion order in the map, use a LinkedHashMap.
- If you need the map to be sorted on keys, use a TreeMap.

Questions

What is the output of the following code fragment?

```
Map<String, String> map = new LinkedHashMap<>();
map.put("123", "John Smith");
map.put("111", "George Smith");
map.put("123", "Steve Yao");
map.put("222", "Steve Yao");
System.out.println("(1) " + map);
System.out.println("(2) " + new TreeMap<String, String>(map));
```

Questions

Write a program that counts the occurrences of words in a text input by the user and displays the words and their occurrences in alphabetical order of the words.