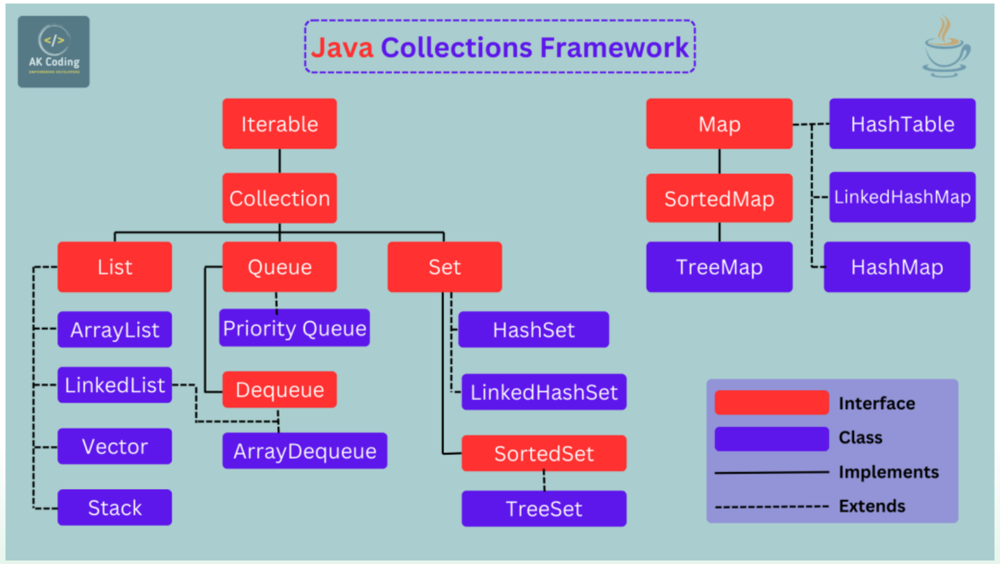
JAVA COLLECTIONS

WHAT ARE COLLECTIONS

Set of interfaces and classes in which data can be stored to be searched and retrieved, the elements stored can be applied functions, ordered and modified inside the collections.

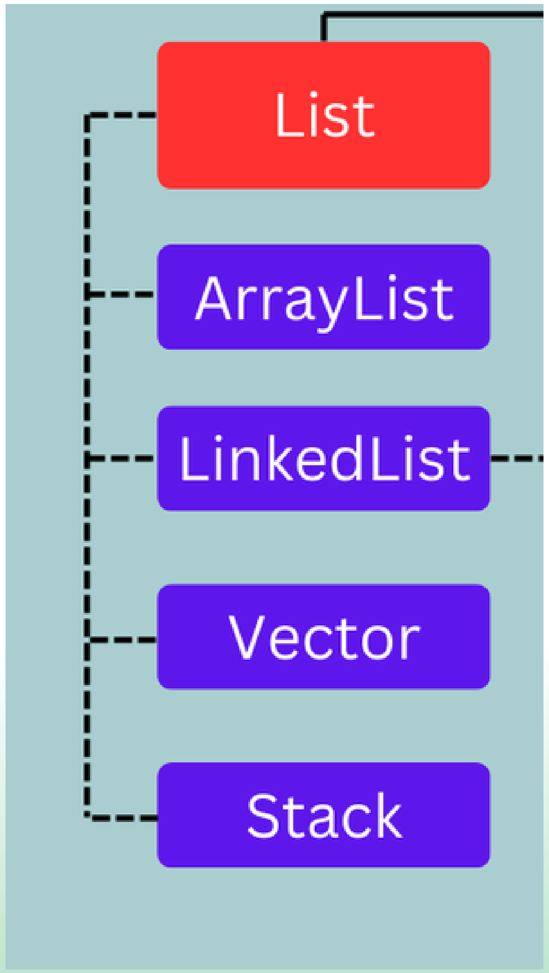
The main methods of the Collection interface are add, remove and size.

To visit each element of a Collection it uses an iterator interface which traverses it sequentially



LIST

List is an interface in which the elements are ordered, which provides a way to get them with a position index, it can also contain duplicates of an element.



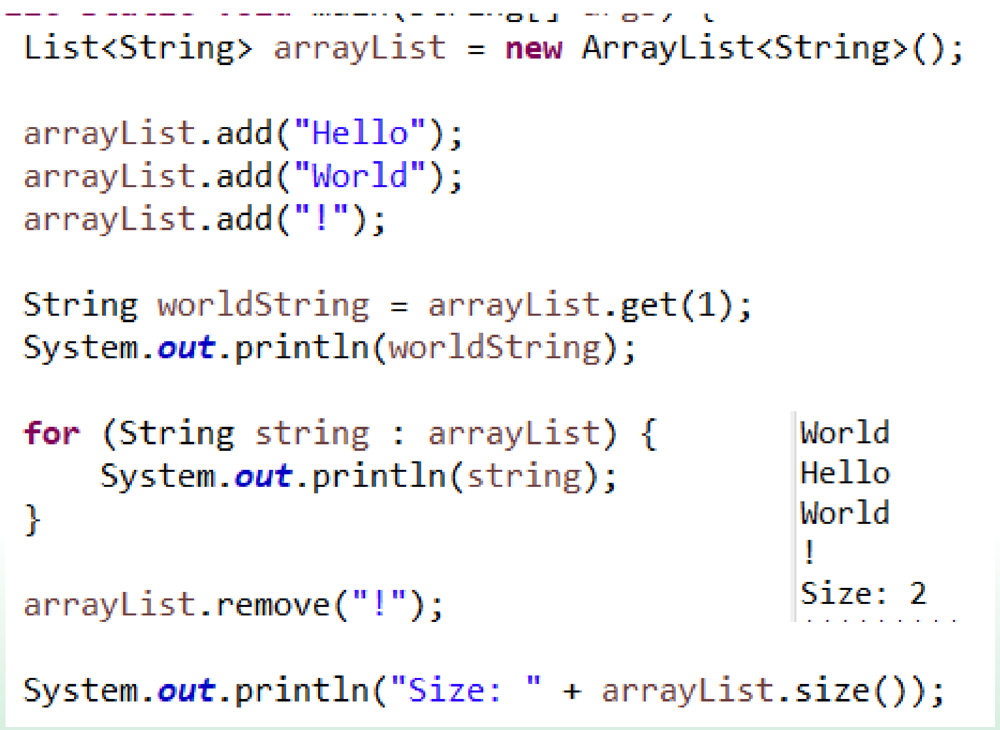
ARRAYLIST

An implementation of the List Interface with dynamic size, making it popular for dynamic data storage.

The sample code shows the main methods of a list, the arrayList class allows for the size to not be specified and the elements added dynamically.

The get method comes from List and retrieves the element in the specified index.

The remove method deletes the first instance of the element specified.



```
List<String> arrayList = new ArrayList<String>();

arrayList.add("Hello");
arrayList.add("World");
arrayList.add("!");

String worldString = arrayList.get(1);
System.out.println(worldString);

for (String string : arrayList) {
    System.out.println(string);
}

arrayList.remove("!");

System.out.println("Size: " + arrayList.size());
```

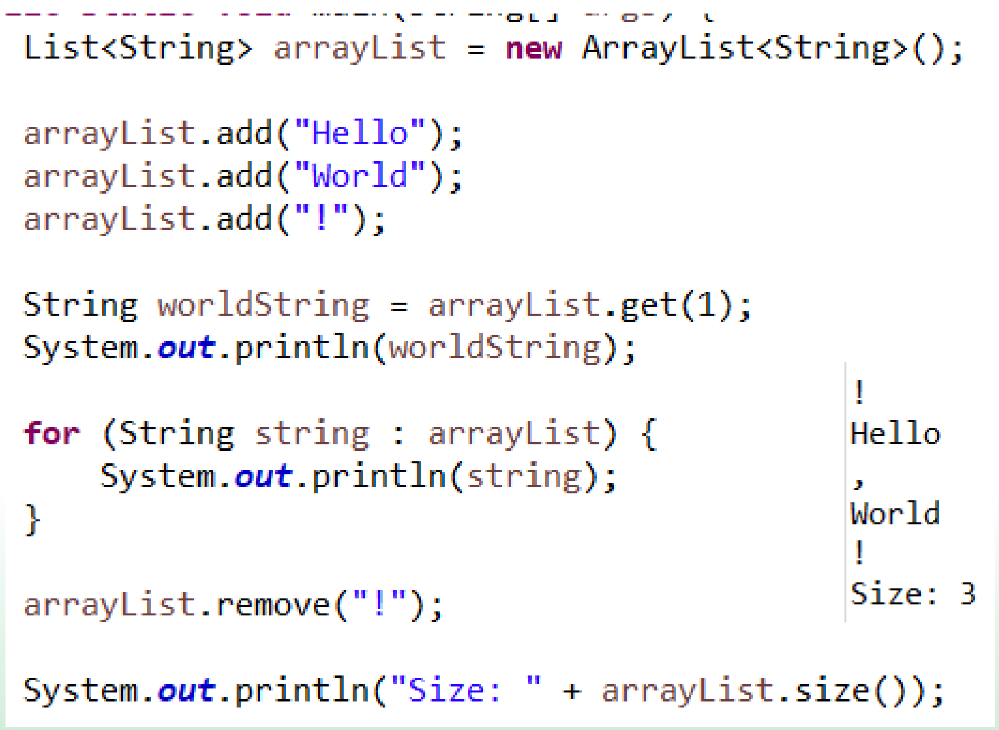
```
World
Hello
World
!
Size: 2
```

The size method returns the number of elements in the array.

LINKEDLIST

An implementation of the List Interface in which the elements are nodes with information of the next and previous elements.

This difference makes it easy to insert elements by updating the references in the nodes.



```java
List<String> arrayList = new ArrayList<String>();

arrayList.add("Hello");
arrayList.add("World");
arrayList.add("!");

String worldString = arrayList.get(1);
System.out.println(worldString);

for (String string : arrayList) {
    System.out.println(string);
}

arrayList.remove("!");

System.out.println("Size: " + arrayList.size());
```

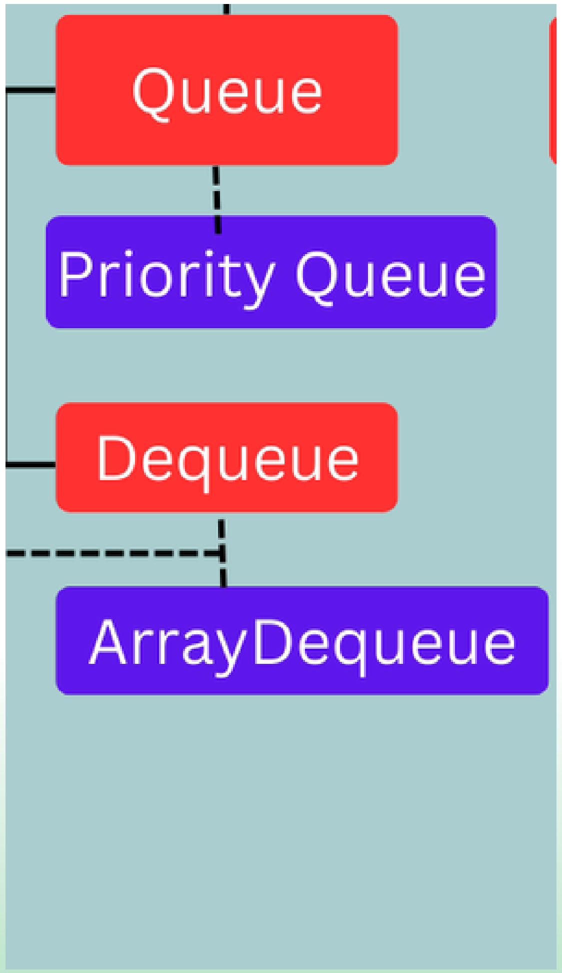
```
!
Hello
,
World
!
Size: 3
```

QUEUE

Queue is an interface in which the first element introduced is the first one out, only allowing that element to be accessed.

We can use poll or remove to get and delete the element from the queue and peek to get the element but not delete it.

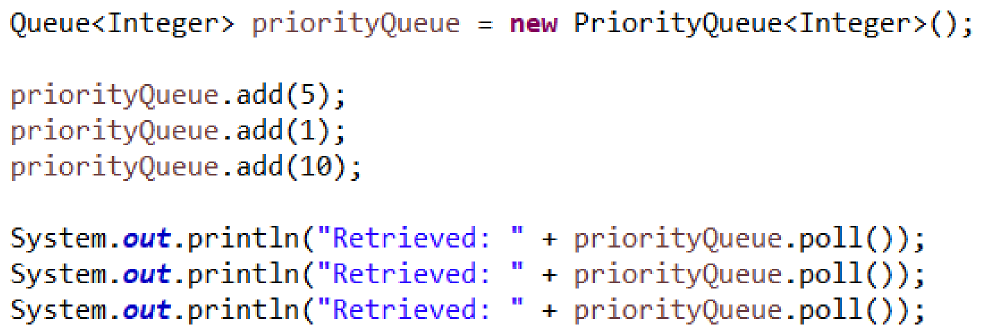
LinkedList is also an implementation of this interface.



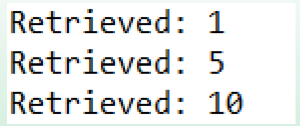
PRIORITYQUEQUE

As the elements are inserted in this collection, they are ordered, it could be by numeric order, alphabetic order or using a specified comparator.

Elements with the highest priority are dequeued first.



```java
Queue<Integer> priorityQueue = new PriorityQueue<Integer>();

priorityQueue.add(5);
priorityQueue.add(1);
priorityQueue.add(10);

System.out.println("Retrieved: " + priorityQueue.poll());
System.out.println("Retrieved: " + priorityQueue.poll());
System.out.println("Retrieved: " + priorityQueue.poll());
```

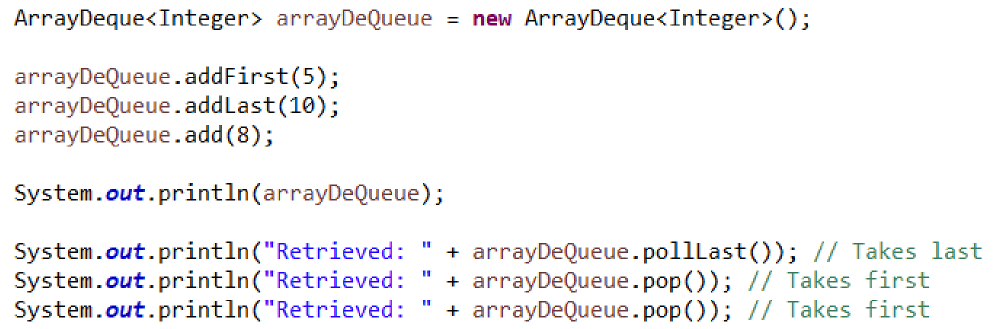


```
Retrieved: 1
Retrieved: 5
Retrieved: 10
```

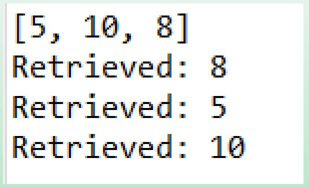
ARRAY DEQUE

It can add and remove elements from the beginning and end of the array, functioning as both a stack and a queue with variable size.

It doesn't allow null elements.



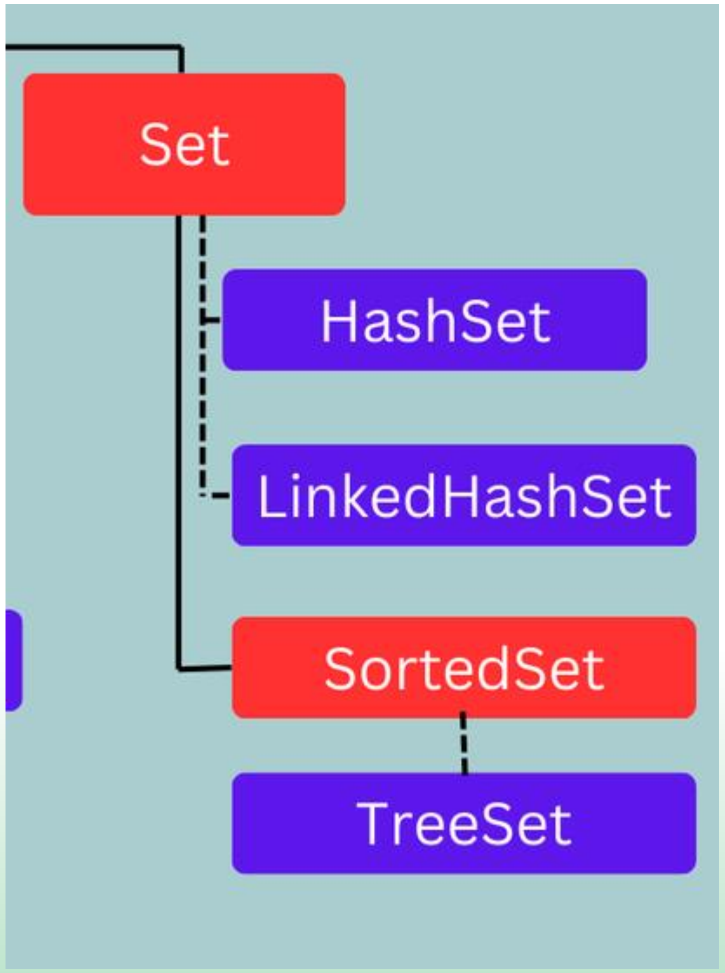
```java
ArrayDeque<Integer> arrayDeQueue = new ArrayDeque<Integer>();

arrayDeQueue.addFirst(5);
arrayDeQueue.addLast(10);
arrayDeQueue.add(8);

System.out.println(arrayDeQueue);

System.out.println("Retrieved: " + arrayDeQueue.pollLast()); // Takes last
System.out.println("Retrieved: " + arrayDeQueue.pop()); // Takes first
System.out.println("Retrieved: " + arrayDeQueue.pop()); // Takes first
```



```
[5, 10, 8]
Retrieved: 8
Retrieved: 5
Retrieved: 10
```

SET

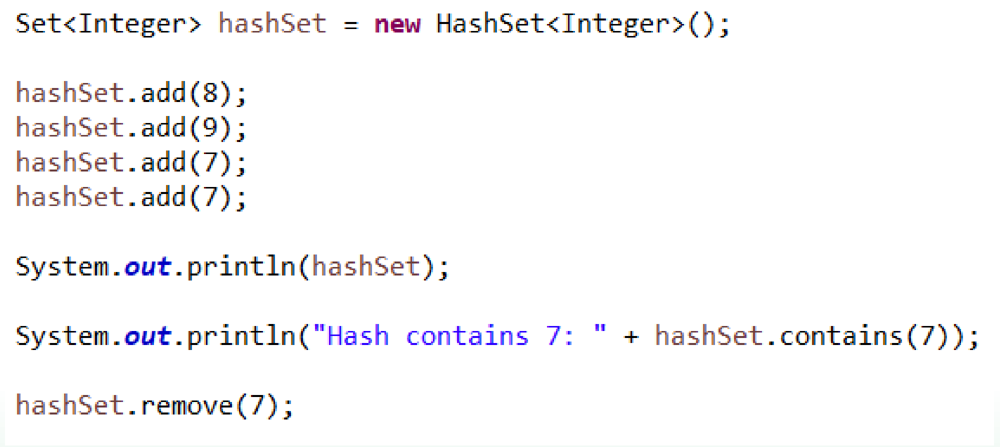
In a set the elements aren’t ordered, so random access is not possible, it also cannot contain duplicates, these characteristics allow for a quick add, remove and search function.

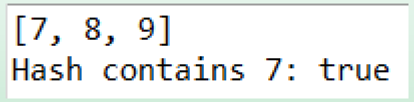


HASHSET

A hashset performs a series of calculations to create a hashcode for an element which determines where to store it in the collection, if two elements happen to get the same hashcode they are stored in the same place and differentiated by the equals function.

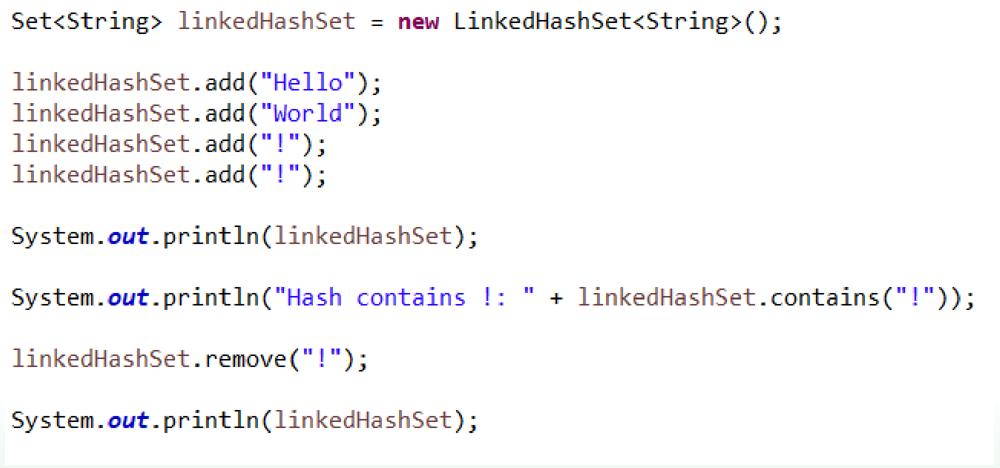
It's used for its performance which depends in how evenly the elements are spread on the set.


```
Set<Integer> hashSet = new HashSet<Integer>();

hashSet.add(8);
hashSet.add(9);
hashSet.add(7);
hashSet.add(7);

System.out.println(hashSet);

System.out.println("Hash contains 7: " + hashSet.contains(7));

hashSet.remove(7);
```

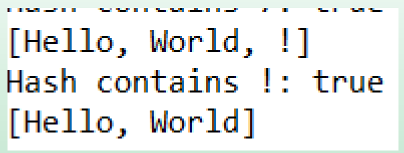

```
[7, 8, 9]
Hash contains 7: true
```

LINKEDHASHSET

Like the other set implementations, it only allows a unique instance of an object, but this collection maintains an order to the elements which are linked to represent this order.



```java
Set<String> linkedHashSet = new LinkedHashSet<String>();

linkedHashSet.add("Hello");
linkedHashSet.add("World");
linkedHashSet.add("!");
linkedHashSet.add("!");

System.out.println(linkedHashSet);

System.out.println("Hash contains !: " + linkedHashSet.contains("!"));

linkedHashSet.remove("!");

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



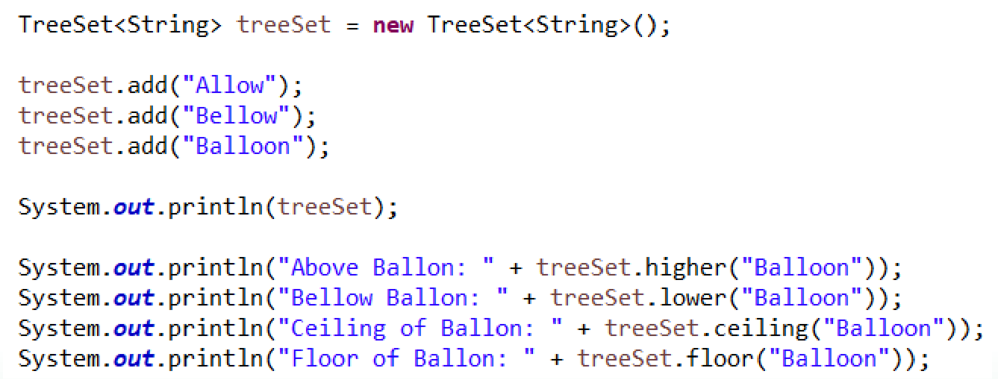
```
[Hello, World, !]
Hash contains !: true
[Hello, World]
```

TREESET

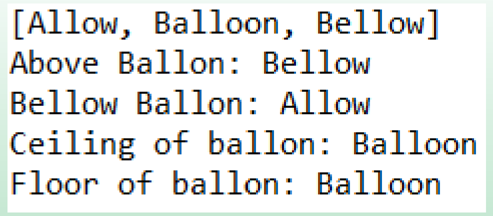
A TreeSet stores elements in the natural order, (alphabetical, ascending, etc.) or according to a comparator.

This allows it to retrieve elements relative to the tree or an element of it (lower <, higher >, ceiling >=, floor <=).

Like all the Sets, it doesn't allow duplicates.



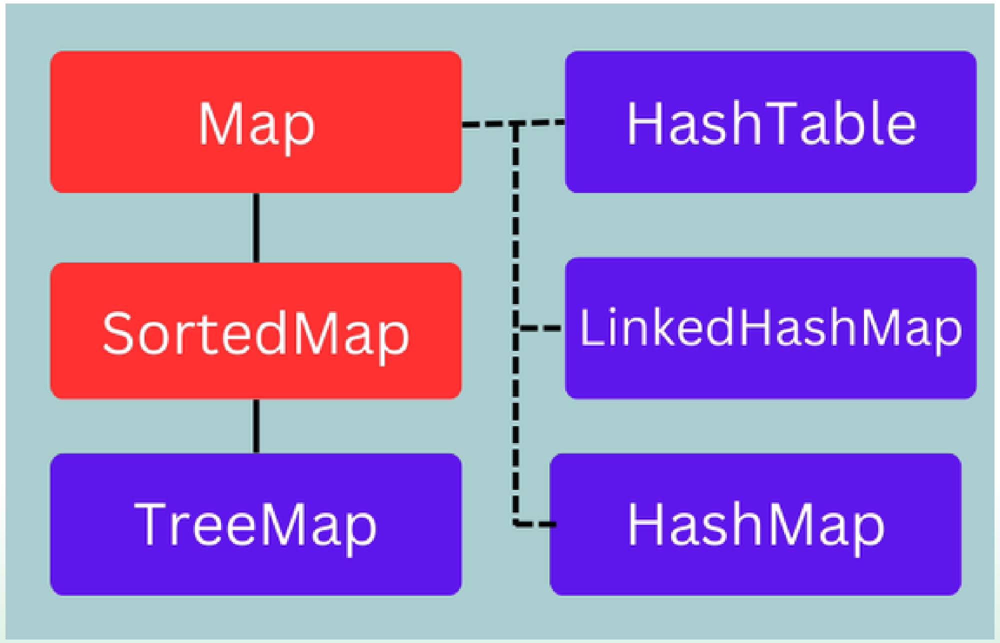
```java
TreeSet<String> treeSet = new TreeSet<String>();

treeSet.add("Allow");
treeSet.add("Bellow");
treeSet.add("Balloon");

System.out.println(treeSet);

System.out.println("Above Ballon: " + treeSet.higher("Balloon"));
System.out.println("Bellow Ballon: " + treeSet.lower("Balloon"));
System.out.println("Ceiling of Ballon: " + treeSet.ceiling("Balloon"));
System.out.println("Floor of Ballon: " + treeSet.floor("Balloon"));
```



```
[Allow, Balloon, Bellow]
Above Ballon: Bellow
Bellow Ballon: Allow
Ceiling of ballon: Balloon
Floor of ballon: Balloon
```

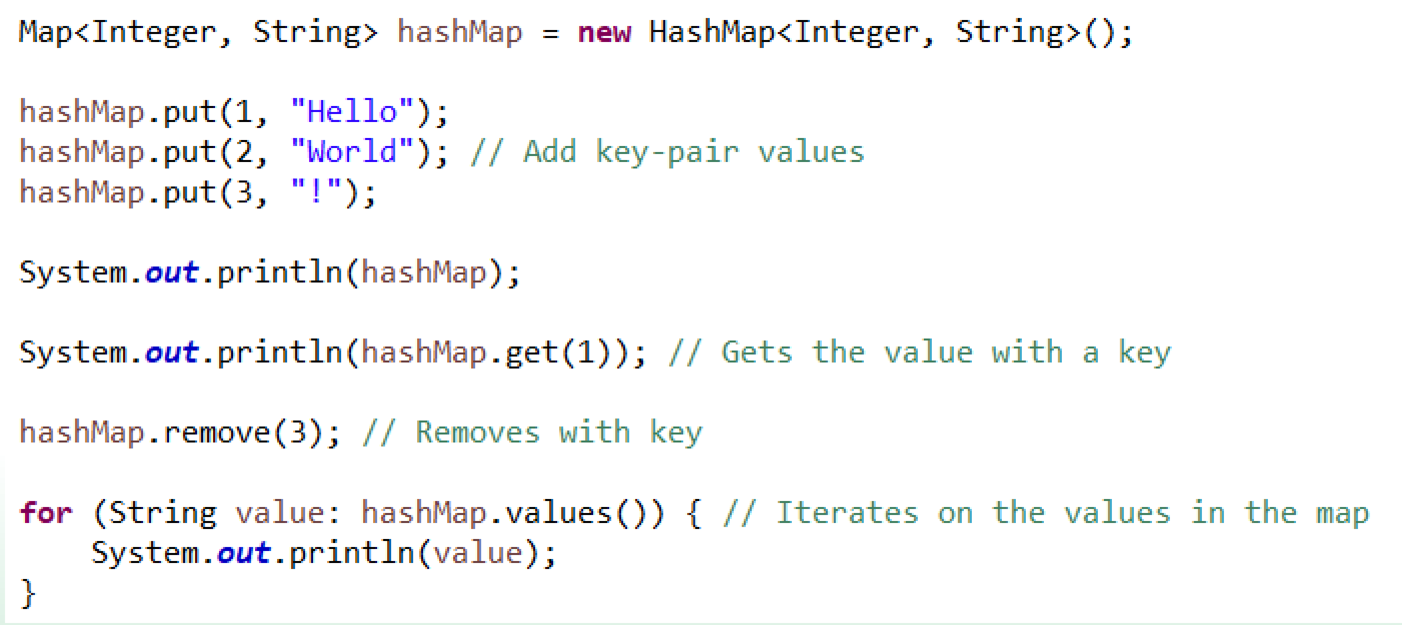
MAPS

Maps do not implement the Collection interface, but still are part of the Collection framework of Java, they consist of a collection of key-pair values where a key is unique in the collection and mapped to just one value.

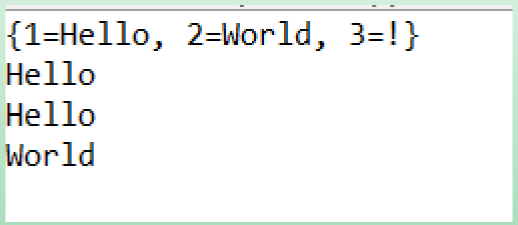


HASHMAP

Simplest implementation of a map, it stores the elements in a hashtable to get the performance benefits of a hashset.



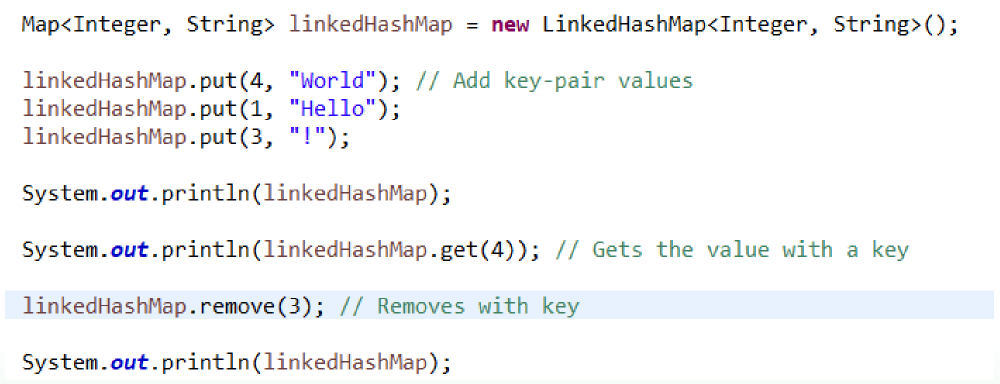
```java
Map<Integer, String> hashMap = new HashMap<Integer, String>();

hashMap.put(1, "Hello");
hashMap.put(2, "World"); // Add key-pair values
hashMap.put(3, "!");

System.out.println(hashMap);

System.out.println(hashMap.get(1)); // Gets the value with a key

hashMap.remove(3); // Removes with key

for (String value: hashMap.values()) { // Iterates on the values in the map
    System.out.println(value);
}
```



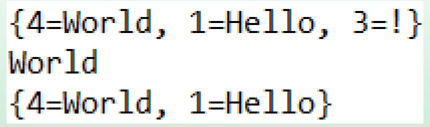
```
{1=Hello, 2=World, 3=!}
Hello
Hello
World
```

LINKEDHASHMAP

Subclass of hashmap that maintains the order based on the insertion of the keys;



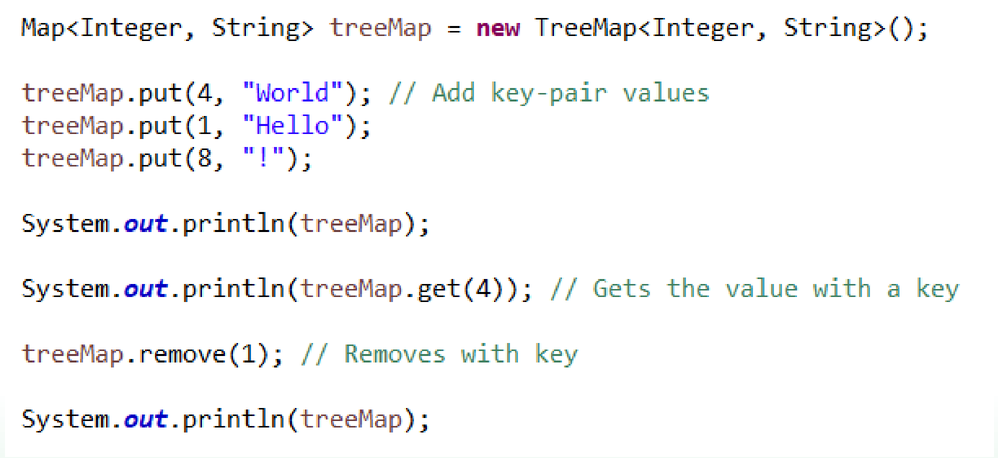
```
Map<Integer, String> linkedHashMap = new LinkedHashMap<Integer, String>();

linkedHashMap.put(4, "World"); // Add key-pair values
linkedHashMap.put(1, "Hello");
linkedHashMap.put(3, "!");

System.out.println(linkedHashMap);

System.out.println(linkedHashMap.get(4)); // Gets the value with a key

linkedHashMap.remove(3); // Removes with key

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



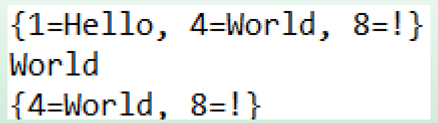
```
{4=World, 1=Hello, 3=!}
World
{4=World, 1=Hello}
```

TREEMAP

Stores the entries in a sorted order based on the keys, this order is the natural order used in previous classes or a custom comparator.



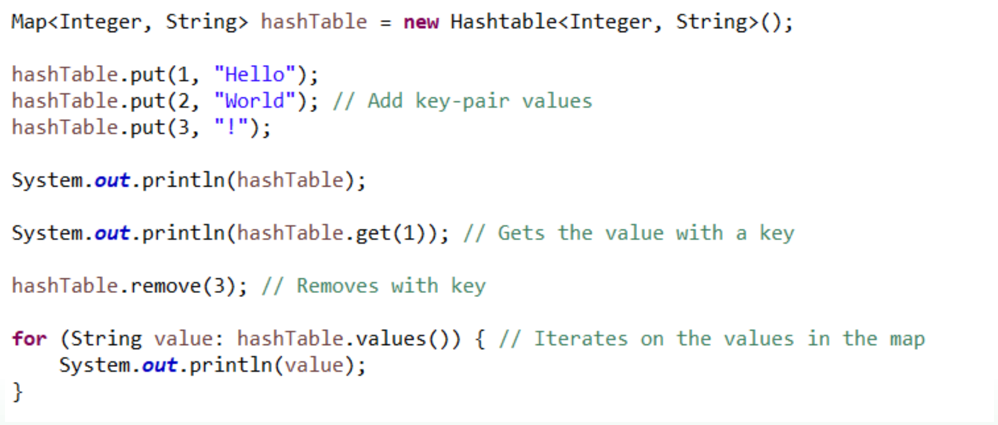
```
Map<Integer, String> treeMap = new TreeMap<Integer, String>();

treeMap.put(4, "World"); // Add key-pair values
treeMap.put(1, "Hello");
treeMap.put(8, "!");

System.out.println(treeMap);

System.out.println(treeMap.get(4)); // Gets the value with a key

treeMap.remove(1); // Removes with key

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



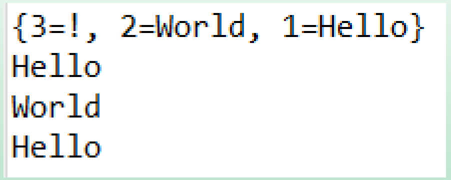
```
{1=Hello, 4=World, 8=!}
World
{4=World, 8=!}
```

HASHTABLE

Legacy class similar to hashmap that doesn't allow null values or keys and it's slower than hashmap due to its synchronization features.



```java
Map<Integer, String> hashTable = new Hashtable<Integer, String>();

hashTable.put(1, "Hello");
hashTable.put(2, "World"); // Add key-pair values
hashTable.put(3, "!");

System.out.println(hashTable);

System.out.println(hashTable.get(1)); // Gets the value with a key

hashTable.remove(3); // Removes with key

for (String value: hashTable.values()) { // Iterates on the values in the map
    System.out.println(value);
}
```



```
{3=!, 2=World, 1=Hello}
Hello
World
Hello
```