What's a view?

The view, or view collection as Java calls it, doesn't store elements, but depends on a backing collection that stores the data elements.

You saw this with the headSet, tailSet and subSet methods on Sets.

You're also very familiar now, with a list backed by an array, a view we get back, when we use the Arrays.asList method, to get an array in the form of a list.

You'll remember when we make changes to that list, the changes are reflected in the underlying array, and vice versa.

The functionality available to us on the list, is limited to features supported by the backing storage, so for a list backed by an array, we can't add or remove elements as an example.

The purpose of view collections

Some of you might be familiar with database views which hide the details of the underlying data structures, to make it easier for clients to use the data.

These view collections serve a similar purpose.

They let us manipulate the collections, without really having to know exact details, about the storage of the data.

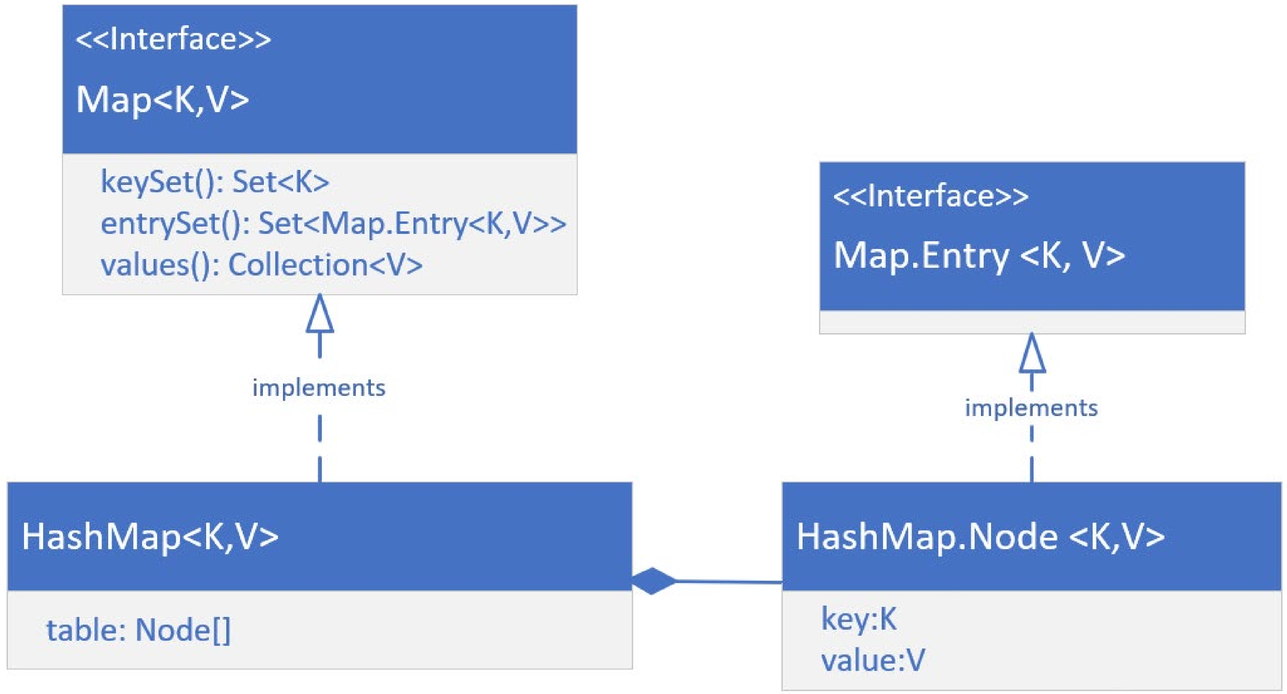
In other words, we don't have to keep learning new methods, to manipulate data.

As long as we can get a collection view of the data, we can use many of the collection methods, to simplify our work.

The HashMap's implementation

On this slide, I'm showing you a high level overview of the structure of the HashMap class.

First, it's important to know that there's a static nested interface on the Map interface, and it's name is Entry.



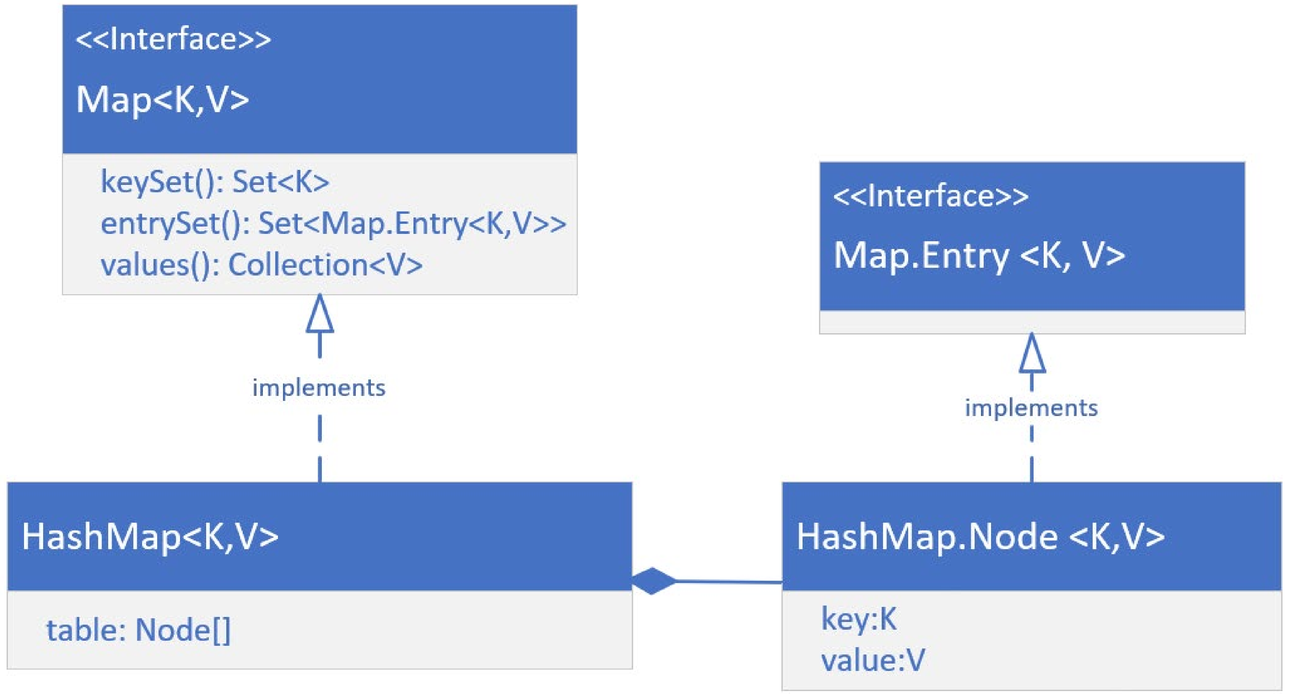
The HashMap's implementation

Concrete classes, implement both the Map interface, and the Map.Entry interface.

The HashMap implements Map, and has a static nested class, Node, that implements the Map.Entry interface.

The HashMap maintains an array of these Nodes, in a field called table, whose size is managed by Java, and whose indices are determined by hashing functions.

For this reason, a HashMap is not ordered.



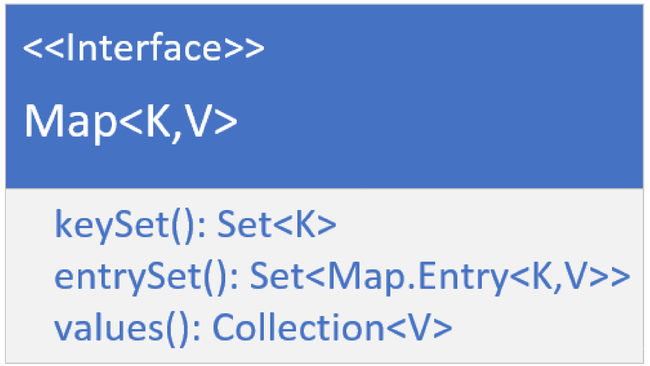
The Map's view collections

I want to focus now on the three view collections we can get from the map, which are the key set, the entry set, and the values.

We know a map has keys, and these can't contain duplicates.

These keys can be retrieved as a Set view, by calling the keySet method on any map object.

Each key value pair is stored as an instance of an Entry, and the combination of the key and value will be unique, because the key is unique.

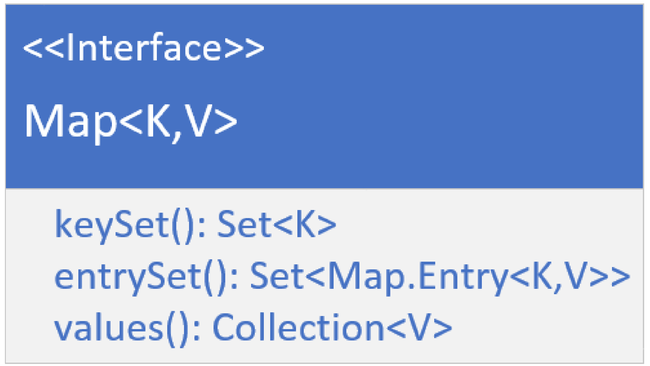


The Map's view collections

A Set view of these entries, or nodes in the case of the HashMap, can be retrieved from the method entrySet.

Finally, values are stored, and referenced by the key, and values can have duplicates, meaning multiple keys could reference a single value.

You can get a Collection view of these, from the values method, on a map instance.



Functionality available to set returned from keySet()

The set returned from the keySet method, is backed by the map.

This means changes to the map are reflected in the set, and vice-versa.

The set supports element removal, which removes the corresponding mapping from the map.

You can use the methods `remove`, `removeAll`, `retainAll`, and clear.

It does not support the `add` or `addAll` operations.