

A decorative graphic on the right side of the page. It features three overlapping circles of varying sizes, each composed of concentric rings in shades of green. Two thin green lines cross the page diagonally, one from the top-left to the bottom-right, and another from the top-right to the bottom-left, intersecting near the center. The circles are positioned in the upper right and lower right areas.

Maps

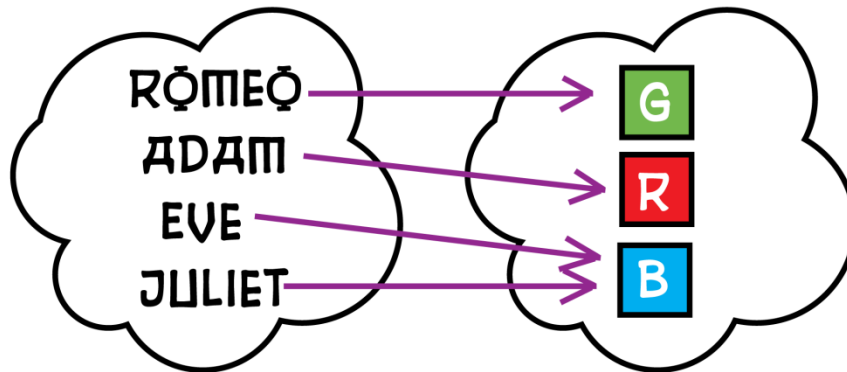
The Map Interface, HashMaps, & TreeMaps

This document contains a description of sets in general, basic operations in the Java 5 Map API, and discussion of HashMap and TreeMap (which implement it).

What is a Map?

A map is a collection data type that maintains **key-to-value** mappings. That means it keeps associations between **keys** and **values**. Both a key and a value can be any object. In math, a map is a function from one set (the **key** set) to another set (the **value** set). Every **key** in a map is **unique/distinct** (no duplicates). It is possible, however, that more than one key can map to the same value. In Java, Map is the name of an interface (`java.util.Map<K,V>`). Two library classes, `HashMap` (`java.util.HashMap<K,V>`) and `TreeMap` (`java.util.TreeMap<K,V>`) implement the interface.

KEYS (KEY SET) VALUES (VALUE SET)



When Might You Use a Map?

- In a large database system, you might have files reside in mass storage. Search is not performed on records, instead a map may be used to link the key with the record's location in storage.

What is in the Map Interface (Java 5)?

- `V put(K key, V value)`
 - Adds an association to the map: associates the specified value with the specified key.
 - If this key was already present in the map, the old value to which it was mapped is replaced.
 - If the key was already in the map, the previous value associated with it is returned.
 - If the key was NOT already in the map, null is returned.
 - If an attempt is made to add the wrong type of object for either the key or value, the method will throw a `ClassCastException`.
- `V get(Object key)`
 - Returns the value to which the map associates the specified key.
 - Returns null if
 - The map contains no mapping for the key
 - The key is explicitly mapped to null
- `V remove(Object key)`
 - Removes the association based on the specified key from the map.
 - Returns the value to which the key was mapped before removal.
 - Returns null if
 - The map contains no mapping for the key
 - The key is explicitly mapped to null

- o boolean containsKey(Object key)
 - Returns true if the map contains an association for the specified key.
- o int size()
 - Returns the number of key-value mappings contained in the map.
- o Set<K>keySet()
 - Returns the set of keys contained in the map.
 - You can then create an iterator for this set, and iterate over it (see sets).

Using the Library Implementations

It is considered good style to store a reference to a HashMap or TreeMap in a variable of type Map, so that if you want to change the implementing class later, it is a simple change of a single word.

```
Map<Type> mapName=new HashMap<Type>();
```

You would replace Type with the data type to be stored in the Map.

Some Example Code

Note – everything except the constructor will look identical for either HashMap or TreeMap.

```
Map<String, Color>favColors=new HashMap<String, Color>();

    //add Associations to the Map
favColors.put("Romeo", Color.GREEN);
favColors.put("Adam", Color.RED);
favColors.put("Eve", Color.BLUE);
favColors.put("Juliet", Color.BLUE);

    //changing an association - just overwrites old association
favColors.put("Juliet", Color.GREEN);

    //accessing a value using the key, print the value
Color evesFav=favColors.get("Eve");
System.out.println(evesFav);

    //Removing an association
favColors.remove("Adam");

    //Getting all keys, iterating through key set to find all associations, print them
Set<String>myKeys=favColors.keySet();
for(String key: myKeys)
{
    Color val=favColors.get(key);
    System.out.println(key + "-->" + val);
}
```

About HashMap...

- Uses Hash table to implement the map.
- No particular order to the elements in the map; ordering may change over time.
- Null values and the null key are permitted.
- Constant time performance O(1) for the get, put, remove, and containsKey operations.
- Rules for Iterators used on the KeySet are same as with Sets in general: If the map is changed once an iterator is constructed without using the iterator's remove method, it will throw a concurrency exception (fail-fast iterator).

About TreeMap...

- Implements Map using a balanced binary search tree.
- Each node in the tree holds a key and reference to the value associated with the key.
- The class guarantees the keys are sorted in ascending order.
- $O(\log(N))$ time guaranteed for the put, get, remove, and containsKey operations.
- Rules for Iterators used on the KeySet are same as with Sets in general: If the map is changed once an iterator is constructed without using the iterator's remove method, it will throw a concurrency exception (fail-fast iterator).