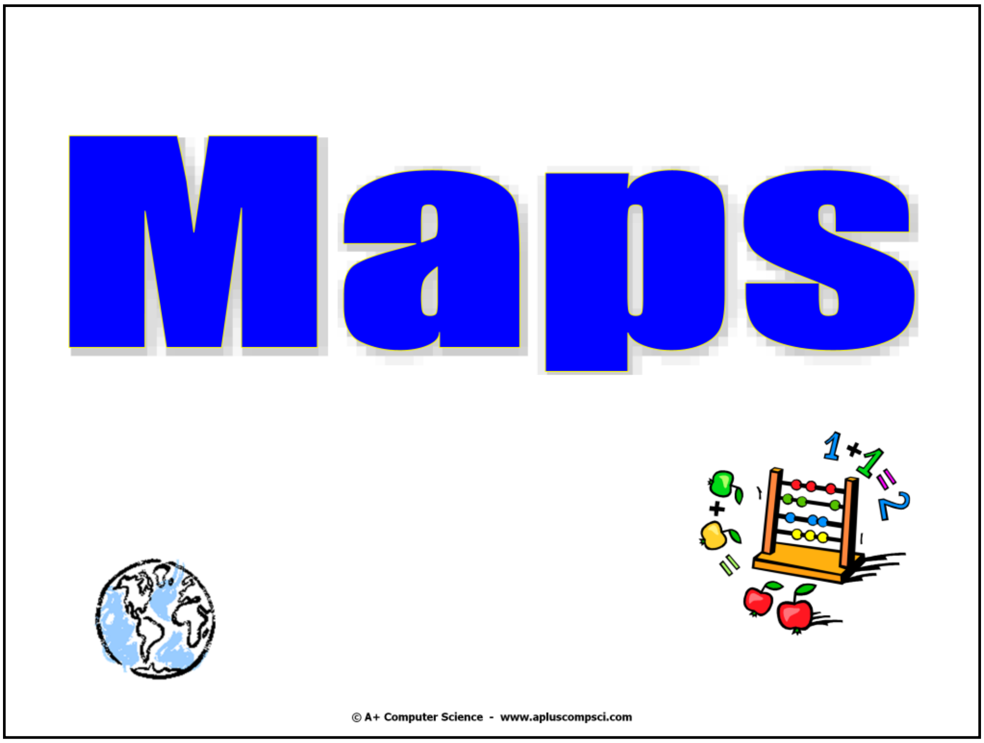
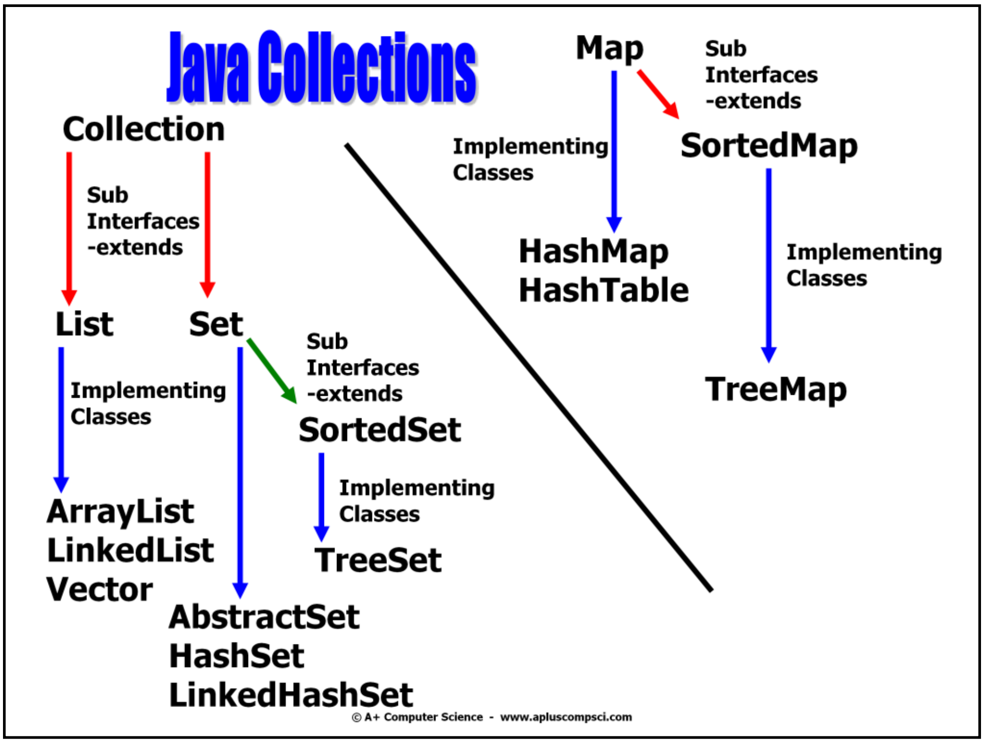


Maps



Java Collections

Collection

Sub Interfaces -extends

List

Set

Implementing Classes

ArrayList
LinkedList
Vector

Sub Interfaces -extends

SortedSet

Implementing Classes

TreeSet

AbstractSet
HashSet
LinkedHashSet

Map

Sub Interfaces -extends

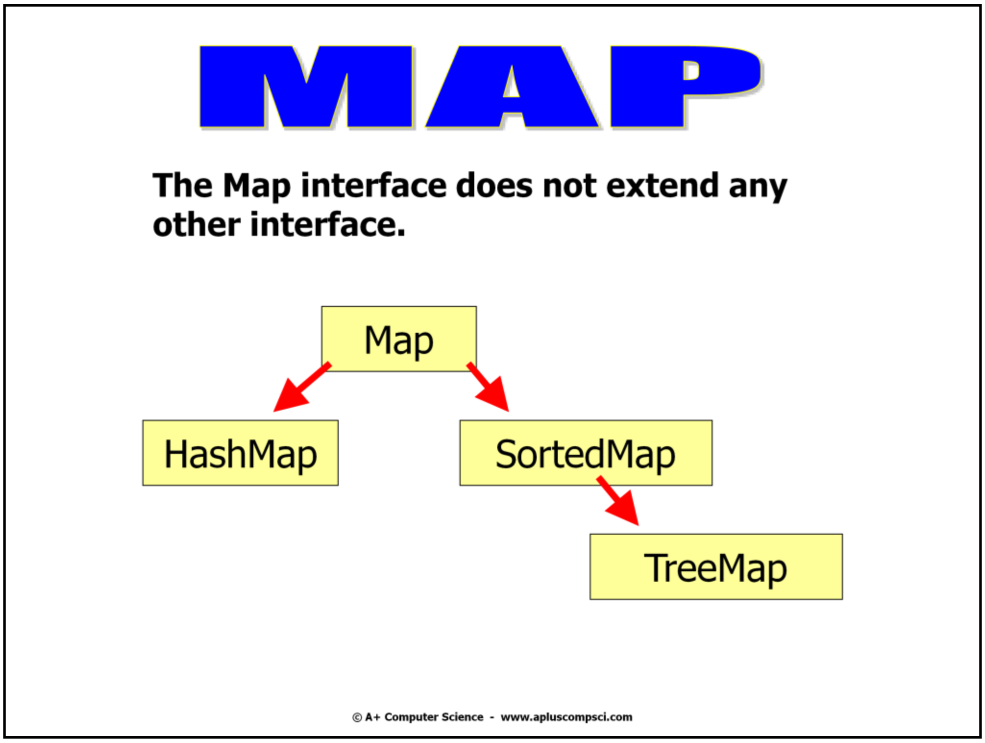
SortedMap

Implementing Classes

HashMap
HashTable

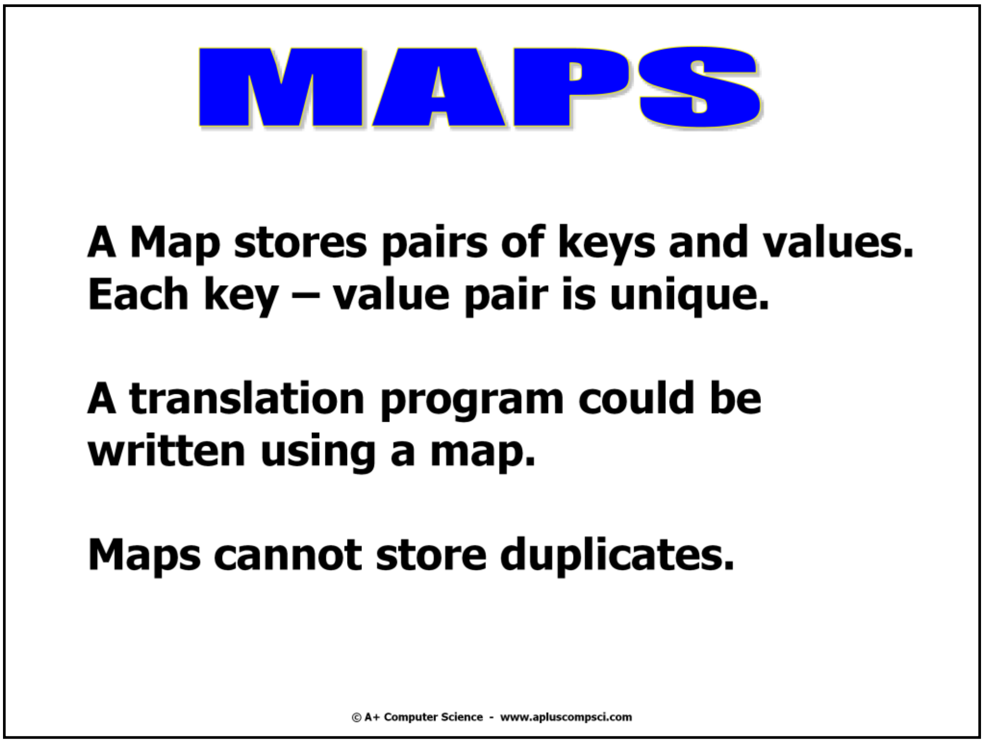
Implementing Classes

TreeMap



MAP

The Map interface does not extend any other interface.

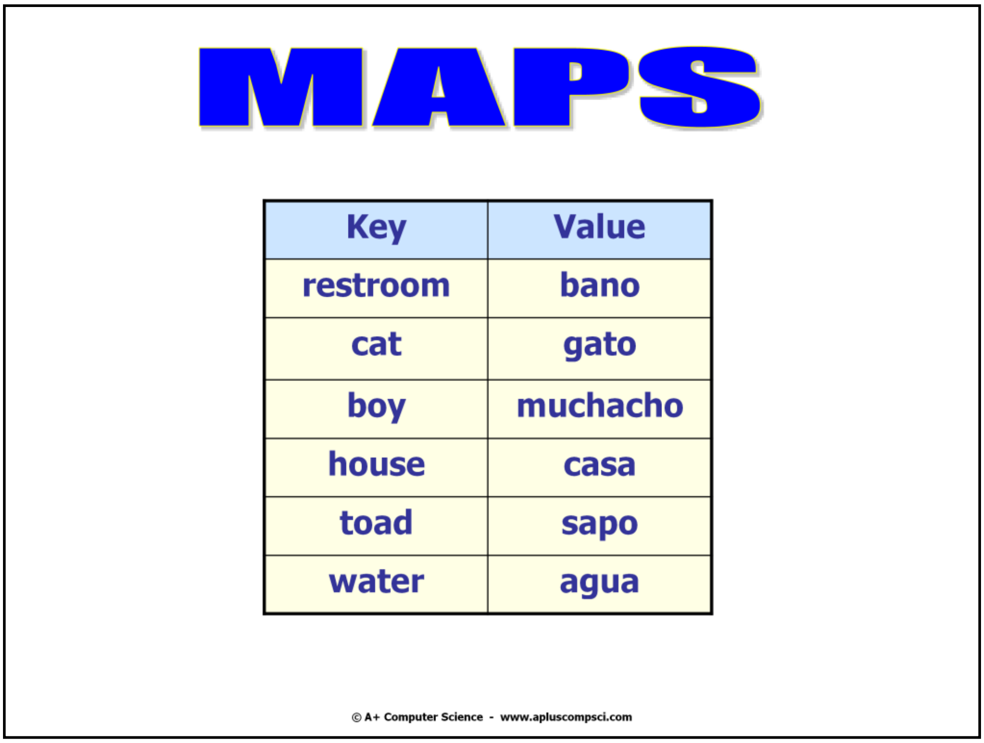


MAPS

A Map stores pairs of keys and values. Each key – value pair is unique.

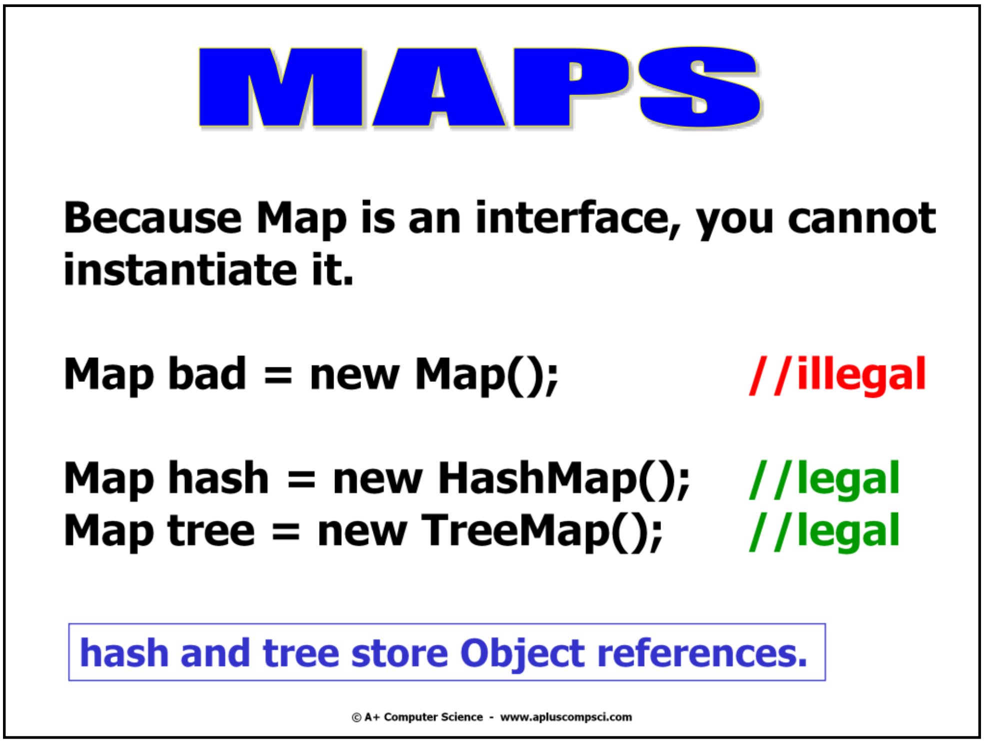
A translation program could be written using a map.

Maps cannot store duplicates.



MAPS

Key	Value
restroom	bano
cat	gato
boy	muchacho
house	casa
toad	sapo
water	agua



MAPS

Because Map is an interface, you cannot instantiate it.

```
Map bad = new Map();              //illegal

Map hash = new HashMap();     //legal
Map tree = new TreeMap();     //legal
```

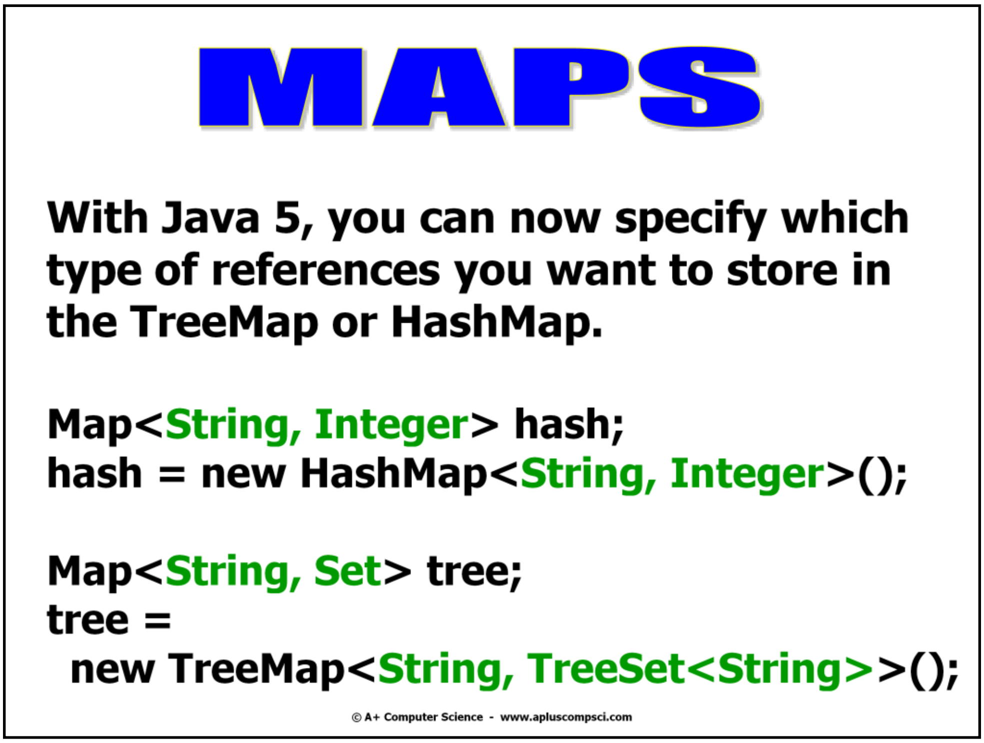
hash and tree store Object references.

Map is an interface; thus, it cannot be instantiated.

```
Map bad = new Map();    //illegal
```

HashMap and TreeMap are children of Map. Map can be used as a reference to any of its children.

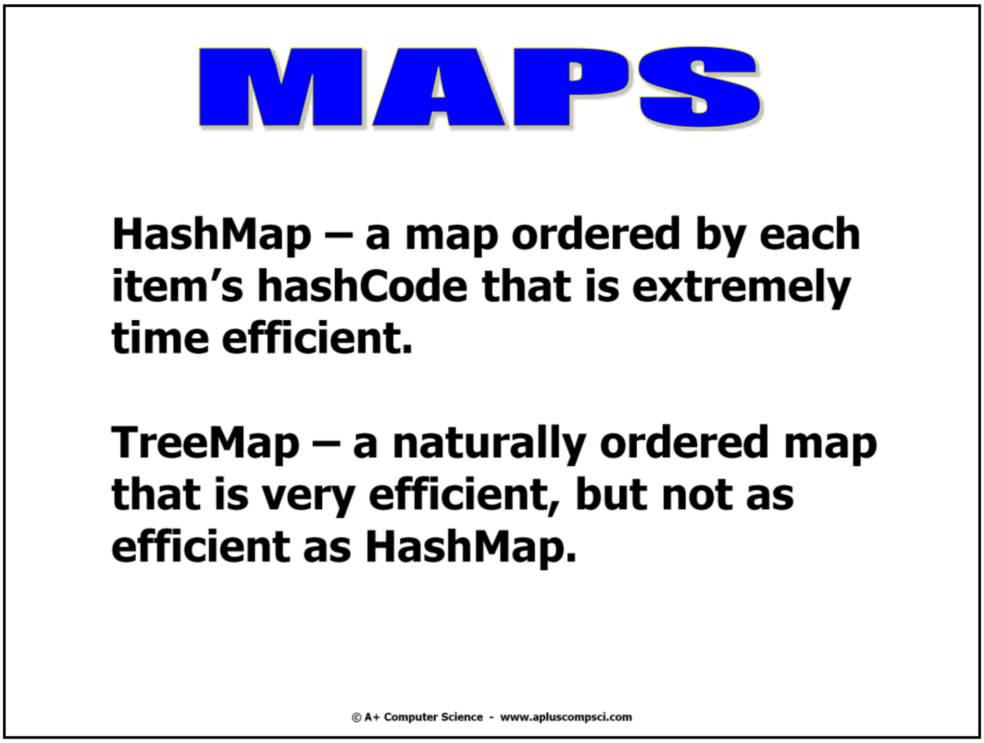
```
Map hash = new HashMap();

Map tree = new TreeMap();
```



MAPS

With Java 5, you can now specify which type of references you want to store in the TreeMap or HashMap.

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

Map<String, Set> tree;
tree =
 new TreeMap<String, TreeSet<String>>();
```



MAPS

HashMap – a map ordered by each item's hashCode that is extremely time efficient.

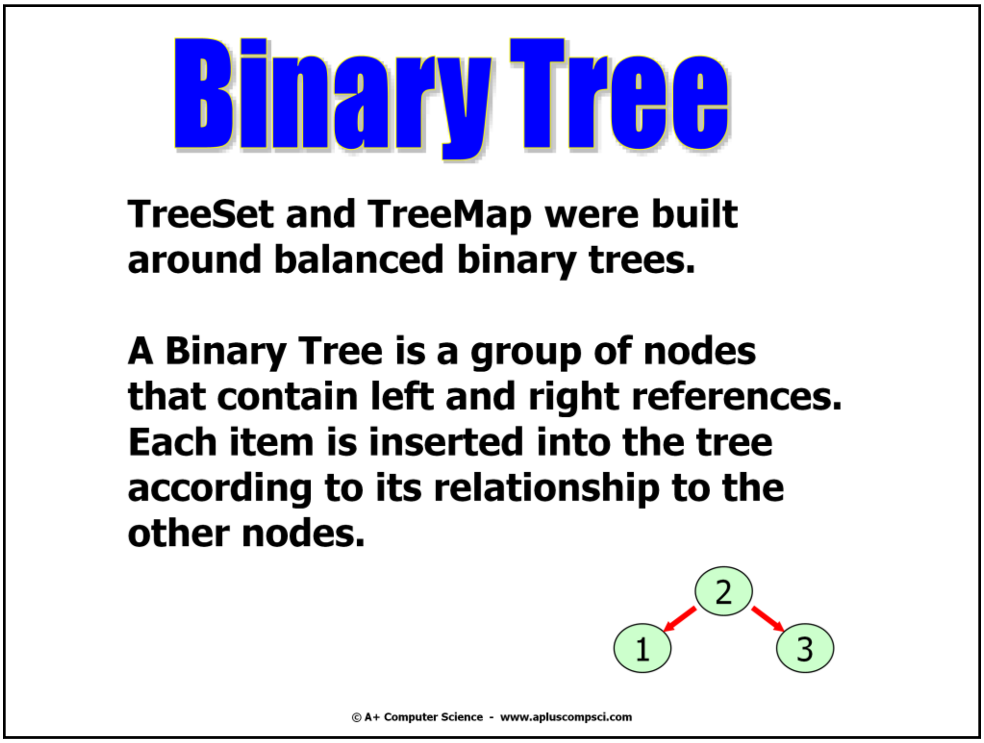
TreeMap – a naturally ordered map that is very efficient, but not as efficient as HashMap.

HashTable

HashSet and HashMap were both created around hash tables.

A hash table is a giant array. Each item is inserted into the array according to a hash formula.

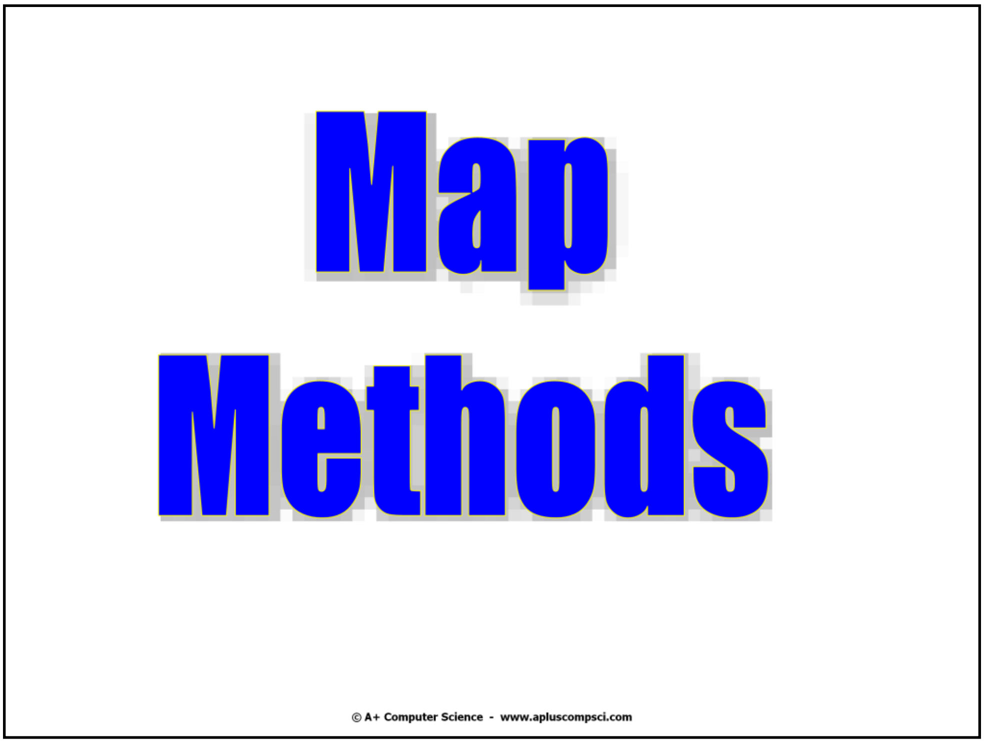
0	1	2	3	4



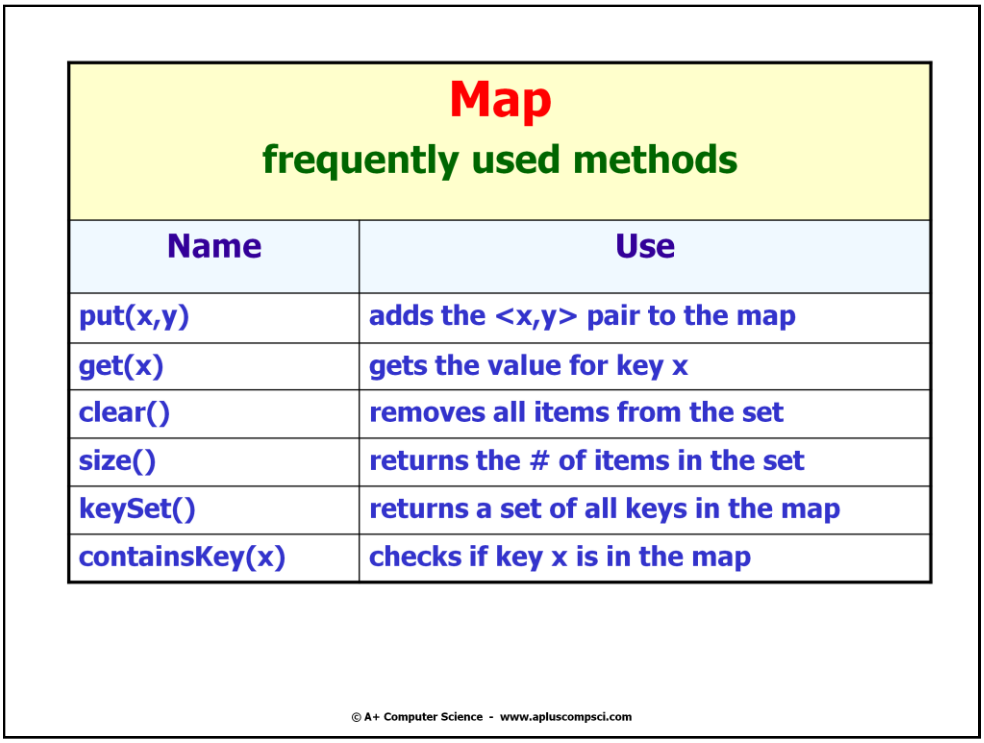
Binary Tree

TreeSet and TreeMap were built around balanced binary trees.

A Binary Tree is a group of nodes that contain left and right references. Each item is inserted into the tree according to its relationship to the other nodes.



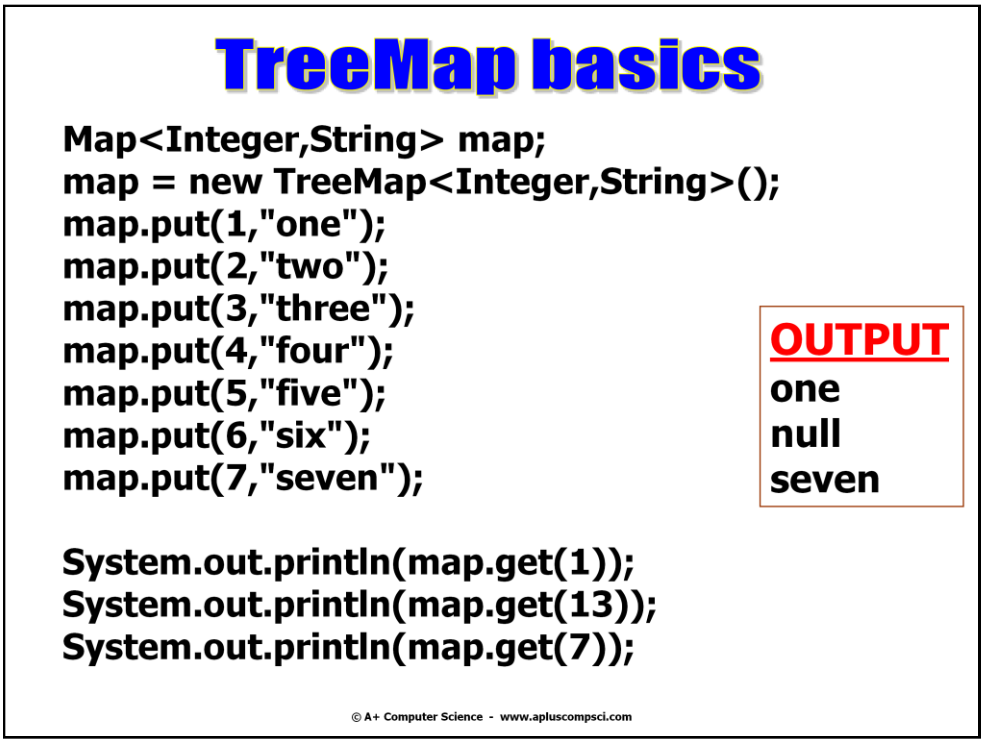
Map Methods



Map

frequently used methods

Name	Use
put(x,y)	adds the <x,y> pair to the map
get(x)	gets the value for key x
clear()	removes all items from the set
size()	returns the # of items in the set
keySet()	returns a set of all keys in the map
containsKey(x)	checks if key x is in the map



TreeMap basics

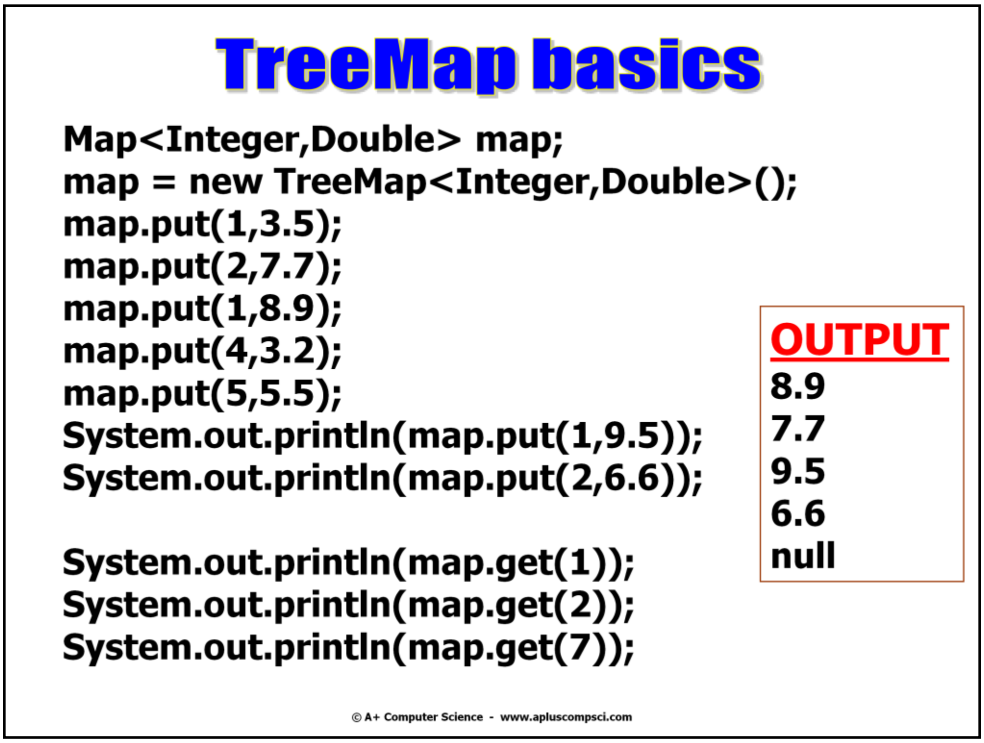
```
Map<Integer,String> map;
map = new TreeMap<Integer,String>();
map.put(1,"one");
map.put(2,"two");
map.put(3,"three");
map.put(4,"four");
map.put(5,"five");
map.put(6,"six");
map.put(7,"seven");

System.out.println(map.get(1));
System.out.println(map.get(13));
System.out.println(map.get(7));
```

OUTPUT

```
one
null
seven
```

The `put()` method is used to put a key,value pair into the map. The `put()` returns a reference to the key that is being replaced.

The `get()` method returns a reference to the value associated with the specified key. If the key specified is not present, the `get()` method returns null.



TreeMap basics

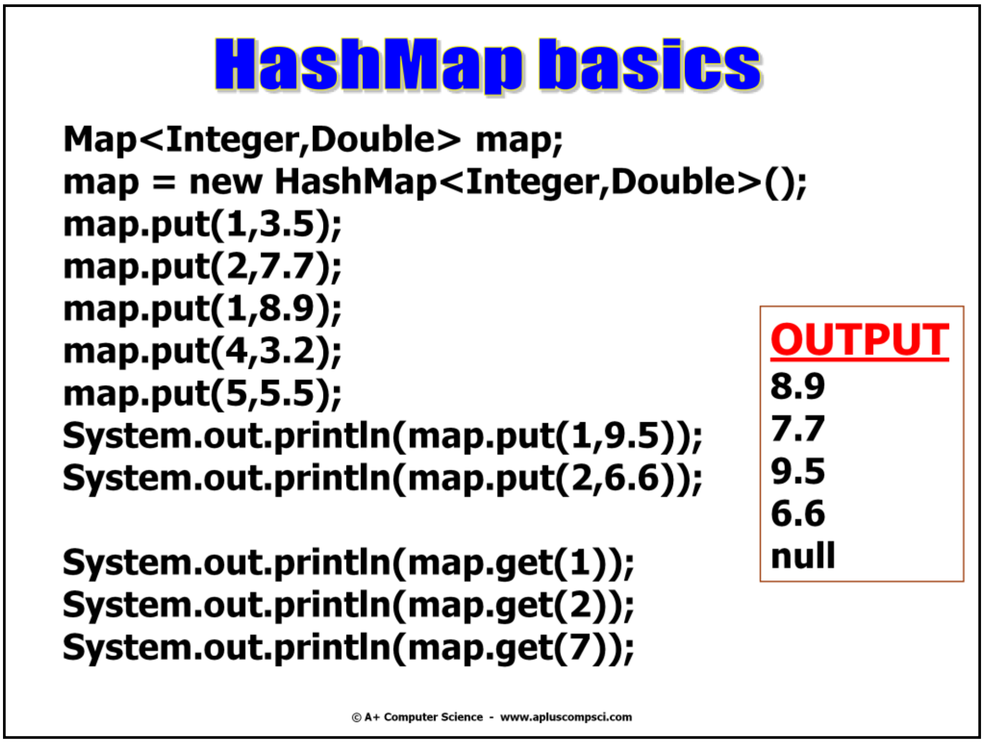
```
Map<Integer,Double> map;
map = new TreeMap<Integer,Double>();
map.put(1,3.5);
map.put(2,7.7);
map.put(1,8.9);
map.put(4,3.2);
map.put(5,5.5);
System.out.println(map.put(1,9.5));
System.out.println(map.put(2,6.6));

System.out.println(map.get(1));
System.out.println(map.get(2));
System.out.println(map.get(7));
```

OUTPUT

```
8.9
7.7
9.5
6.6
null
```

The `put()` method is used to put a key,value pair into the map. The `put()` returns a reference to the key that is being replaced. If the key, value pair was not present, `null` is returned.

The `get()` method returns a reference to the value associated with the specified key. If the key specified is not present, the `get()` method returns null.



HashMap basics

```
Map<Integer,Double> map;
map = new HashMap<Integer,Double>();
map.put(1,3.5);
map.put(2,7.7);
map.put(1,8.9);
map.put(4,3.2);
map.put(5,5.5);
System.out.println(map.put(1,9.5));
System.out.println(map.put(2,6.6));

System.out.println(map.get(1));
System.out.println(map.get(2));
System.out.println(map.get(7));
```

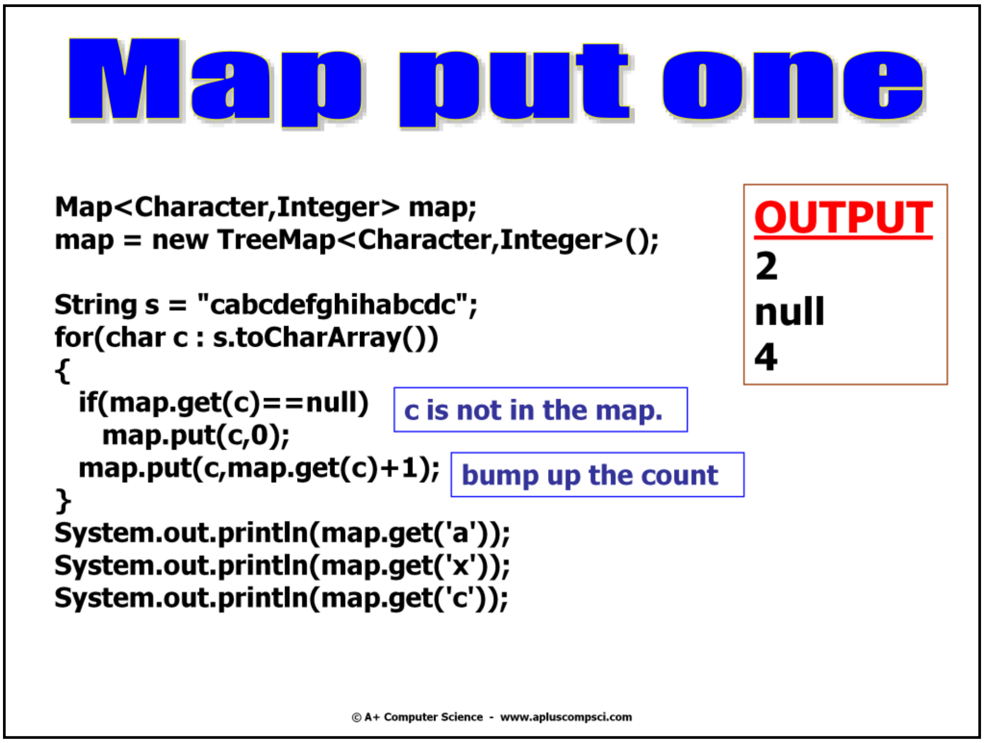
OUTPUT

```
8.9
7.7
9.5
6.6
null
```

The `put()` method is used to put a key,value pair into the map. The `put()` returns a reference to the key that is being replaced. If the key, value pair was not present, `null` is returned.

The `get()` method returns a reference to the value associated with the specified key. If the key specified is not present, the `get()` method returns null.

open
basicmapone.java
basicmaptwo.java
basicmapthree.java



Map put one

```
Map<Character,Integer> map;
map = new TreeMap<Character,Integer>();

String s = "cabcdefghihabcdc";
for(char c : s.toCharArray())
{
  if(map.get(c)==null)
    map.put(c,0);
  map.put(c,map.get(c)+1);
}
System.out.println(map.get('a'));
System.out.println(map.get('x'));
System.out.println(map.get('c'));
```

c is not in the map.

bump up the count

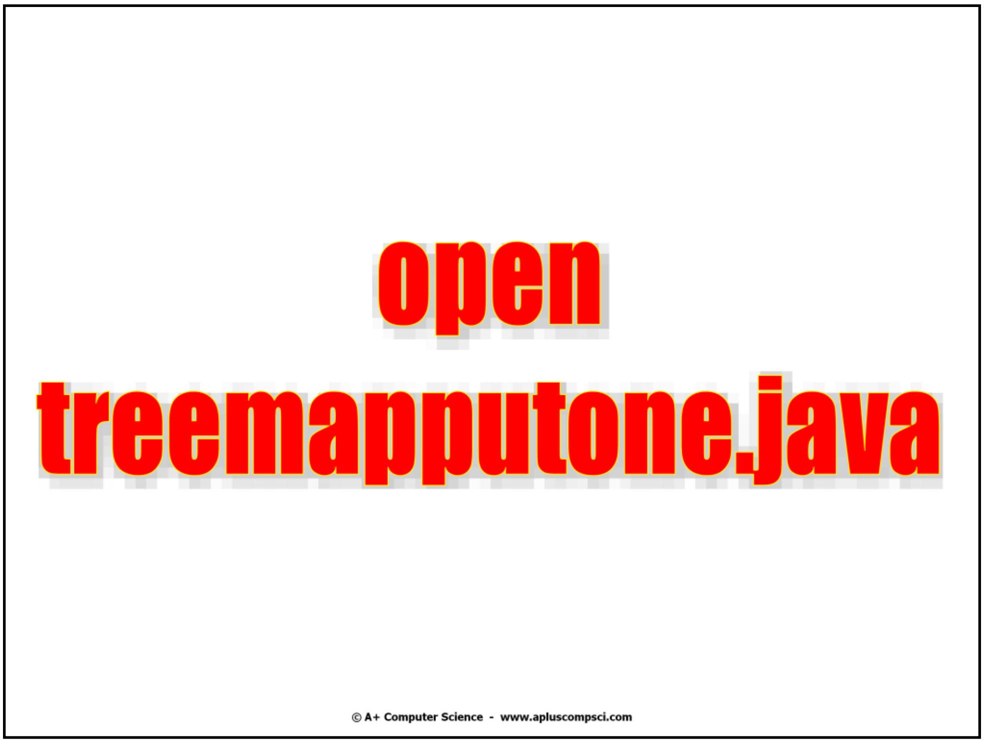
OUTPUT

```
2
null
4
```

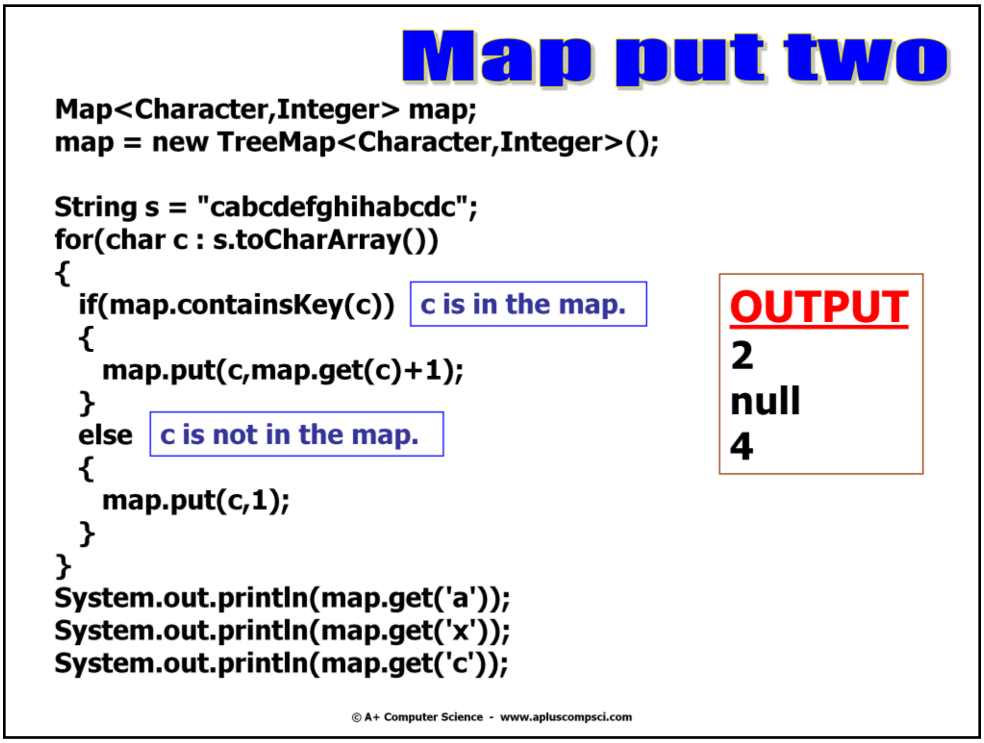
© A+ Computer Science - www.apluscompsci.com

In the code above, the loop iterates over the String s one character at a time. The if checks to see if the current character c is present. If the char is present, the count value is increased by one. If char is not present, the char is put in the map with the value 1.

Because `get()` returns `null` for a key that is not present, the return value for `get()` can be used to determine if a key is present. A `null` return for `get()` indicates that the map does not contain that key.



open treemapputone.java



Map put two

```
Map<Character,Integer> map;
map = new TreeMap<Character,Integer>();

String s = "cabcdefghihabcdc";
for(char c : s.toCharArray())
{
  if(map.containsKey(c))
  {
    map.put(c,map.get(c)+1);
  }
  else
  {
    map.put(c,1);
  }
}
System.out.println(map.get('a'));
System.out.println(map.get('x'));
System.out.println(map.get('c'));
```

c is in the map.

c is not in the map.

OUTPUT

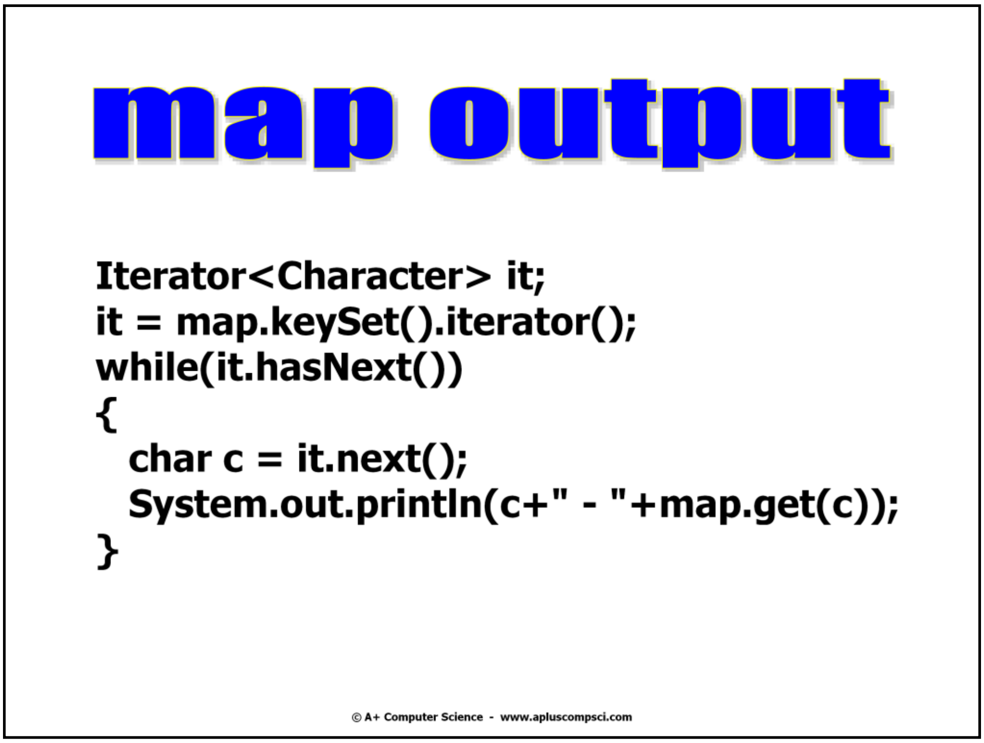
```
2
null
4
```

© A+ Computer Science - www.apluscompsci.com

In the code above, the loop iterates over the String s one character at a time. The if checks to see if the current character c is present. If the char is present, the count value is increased by one. If char is not present, the char is put in the map with the value 1.

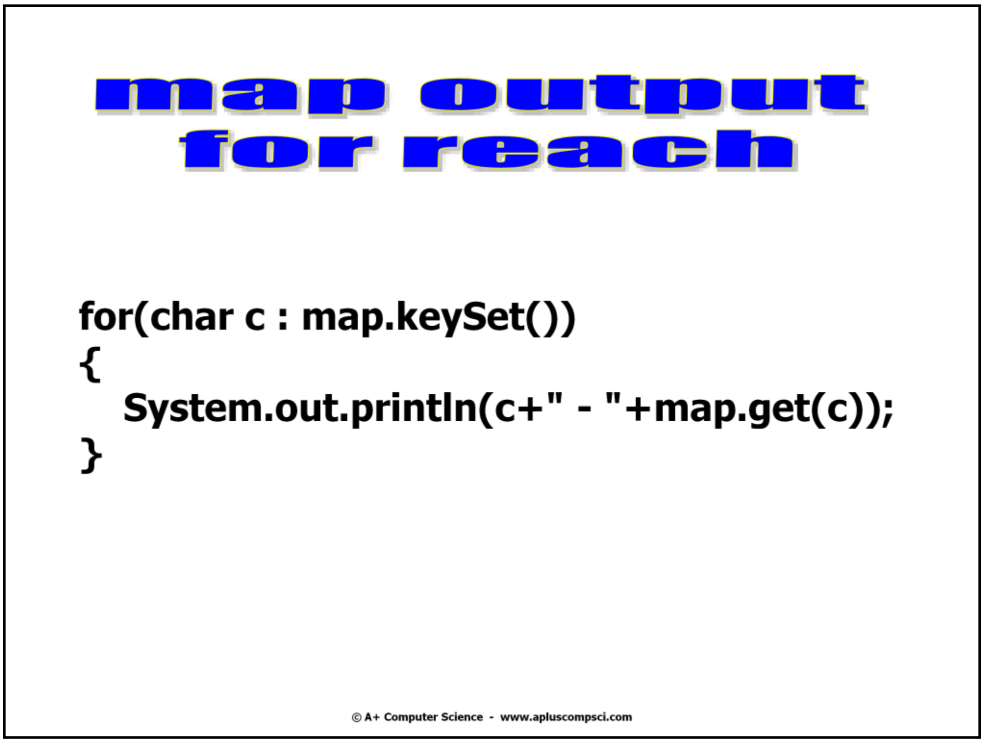
Because `containsKey()` returns `false` for a key that is not present, the return value for `containsKey()` can be used to determine if a key is present. A `false` return for `containsKey()` indicates that the map does not contain that key.

open treemapputtwo.java



map output

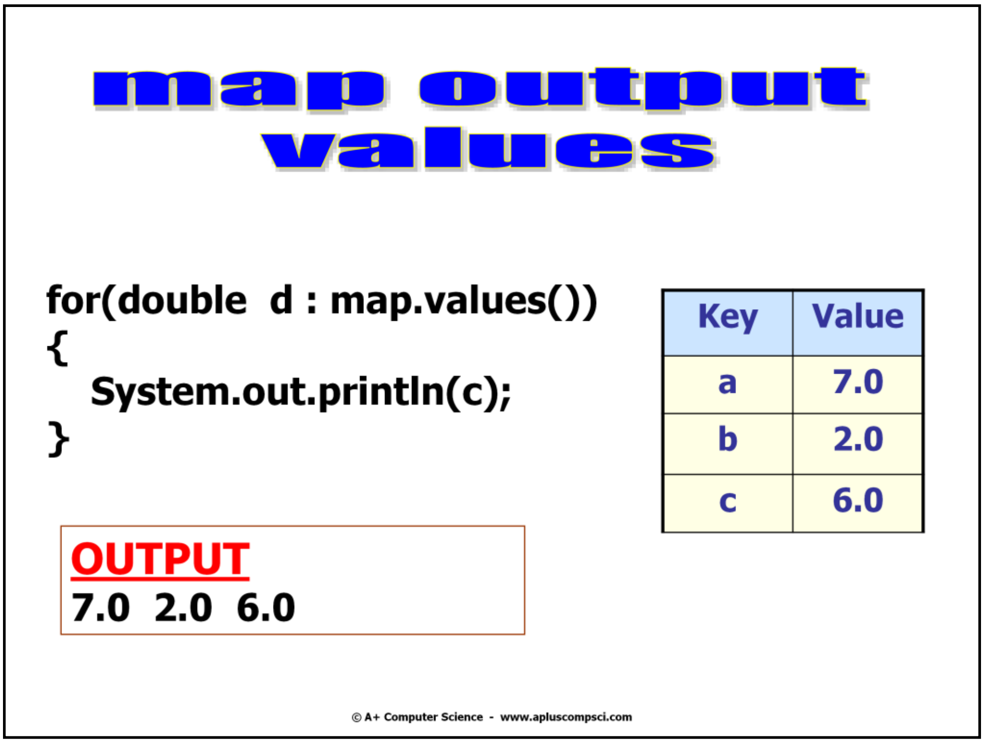
```
Iterator<Character> it;
it = map.keySet().iterator();
while(it.hasNext())
{
  char c = it.next();
  System.out.println(c+" - "+map.get(c));
}
```



map output for reach

```
for(char c : map.keySet())
{
   System.out.println(c+" - "+map.get(c));
}
```

Open treemapoutput.java treemapoutputforeach.java



map output values

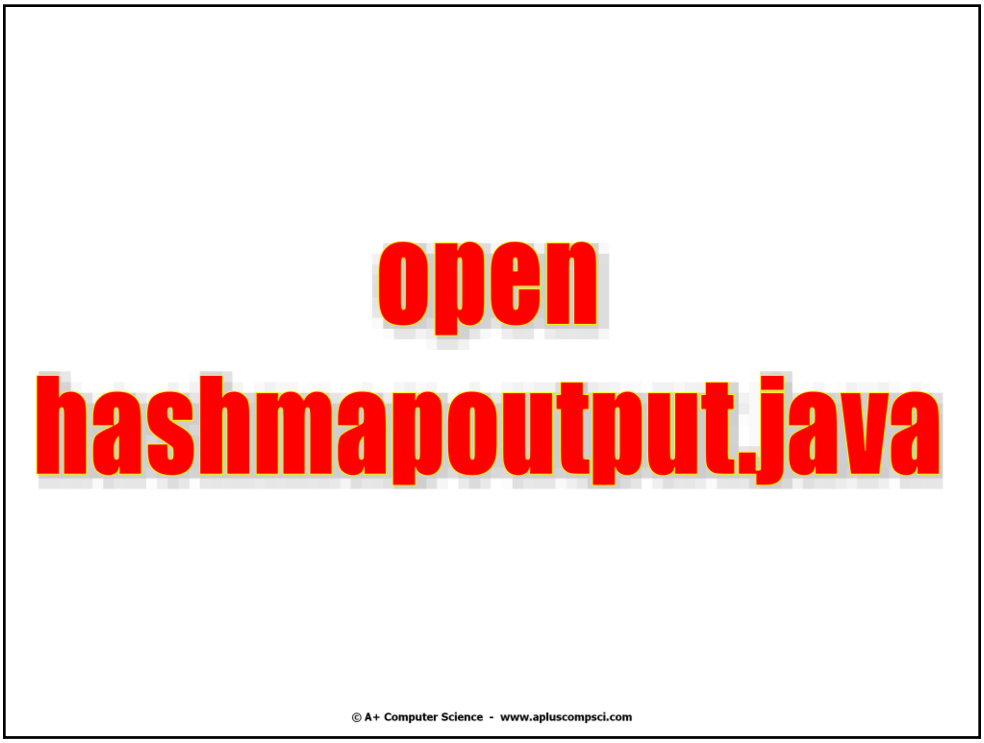
```
for(double  d : map.values())
{
   System.out.println(c);
}
```

Key	Value
a	7.0
b	2.0
c	6.0

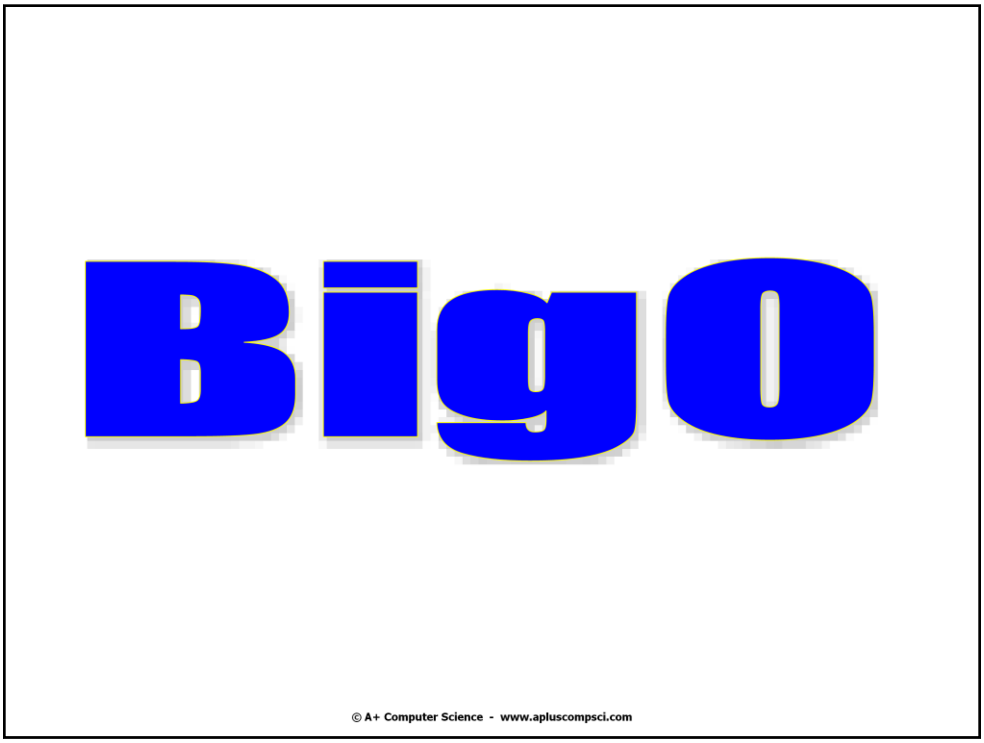
OUTPUT

```
7.0  2.0  6.0
```

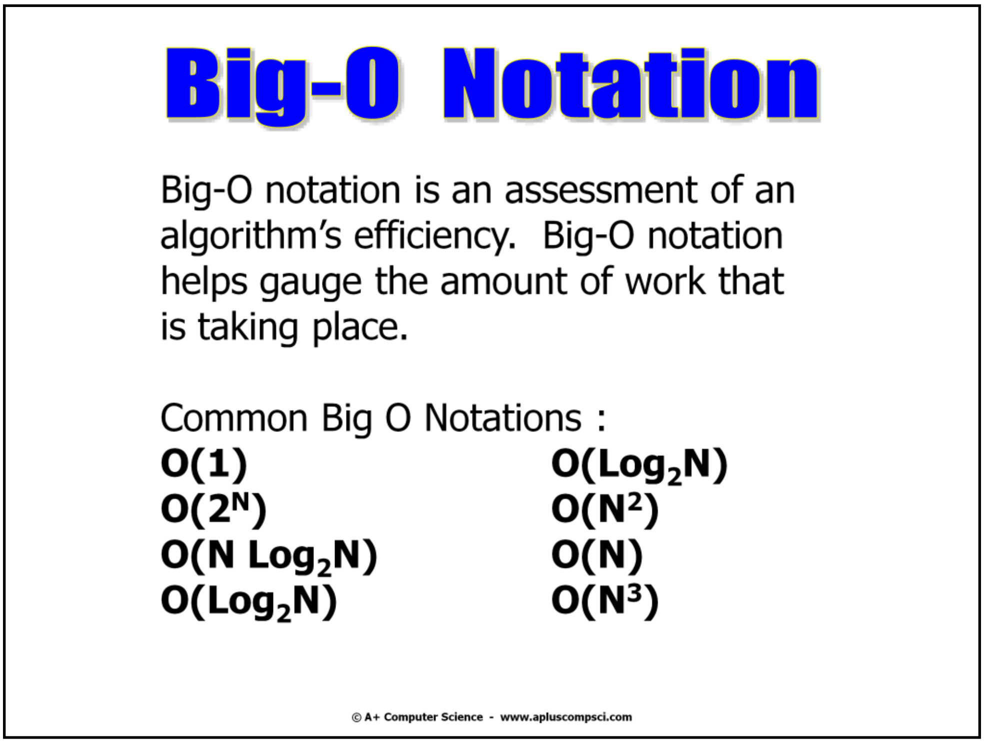
Open
treemapoutputvalues.java



open hashmapoutput.java



BigO

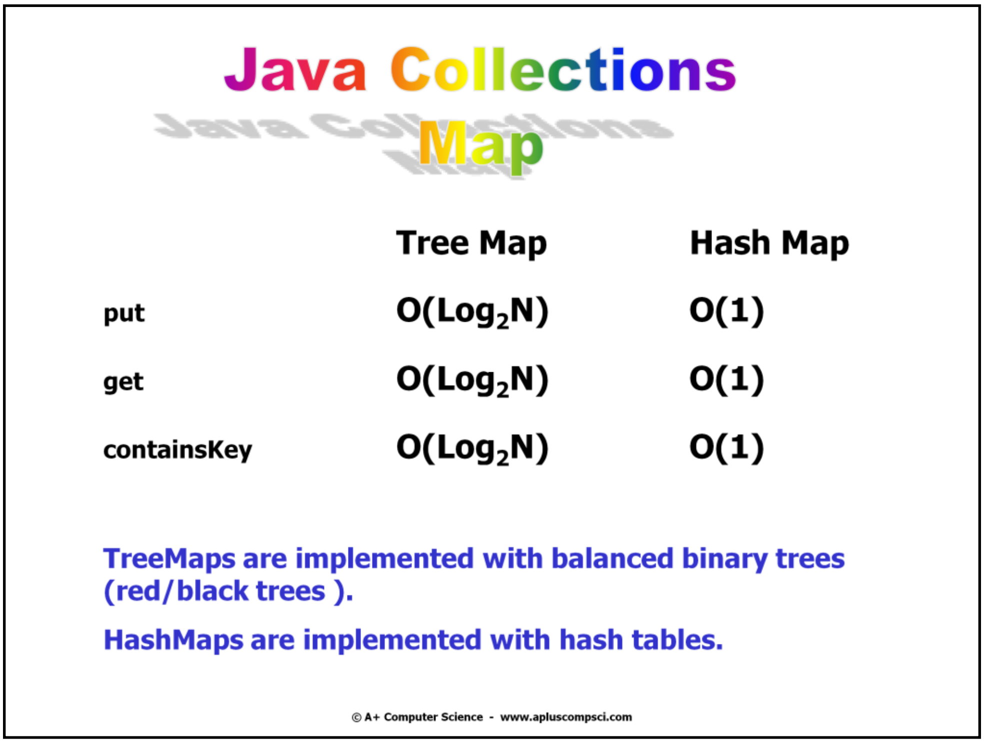


Big-O Notation

Big-O notation is an assessment of an algorithm's efficiency. Big-O notation helps gauge the amount of work that is taking place.

Common Big O Notations :

$O(1)$
$O(2^N)$
$O(N \text{ Log}_2N)$
$O(\text{Log}_2N)$
$O(\text{Log}_2N)$
$O(N^2)$
$O(N)$
$O(N^3)$



Java Collections
Map

	Tree Map	Hash Map
put	O(Log_2N)	O(1)
get	O(Log_2N)	O(1)
containsKey	O(Log_2N)	O(1)

TreeMaps are implemented with balanced binary trees (red/black trees).

HashMaps are implemented with hash tables.

Start work on the labs