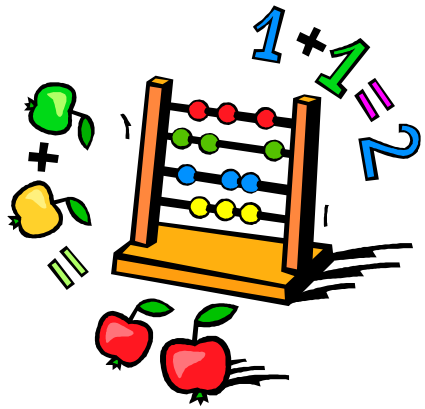
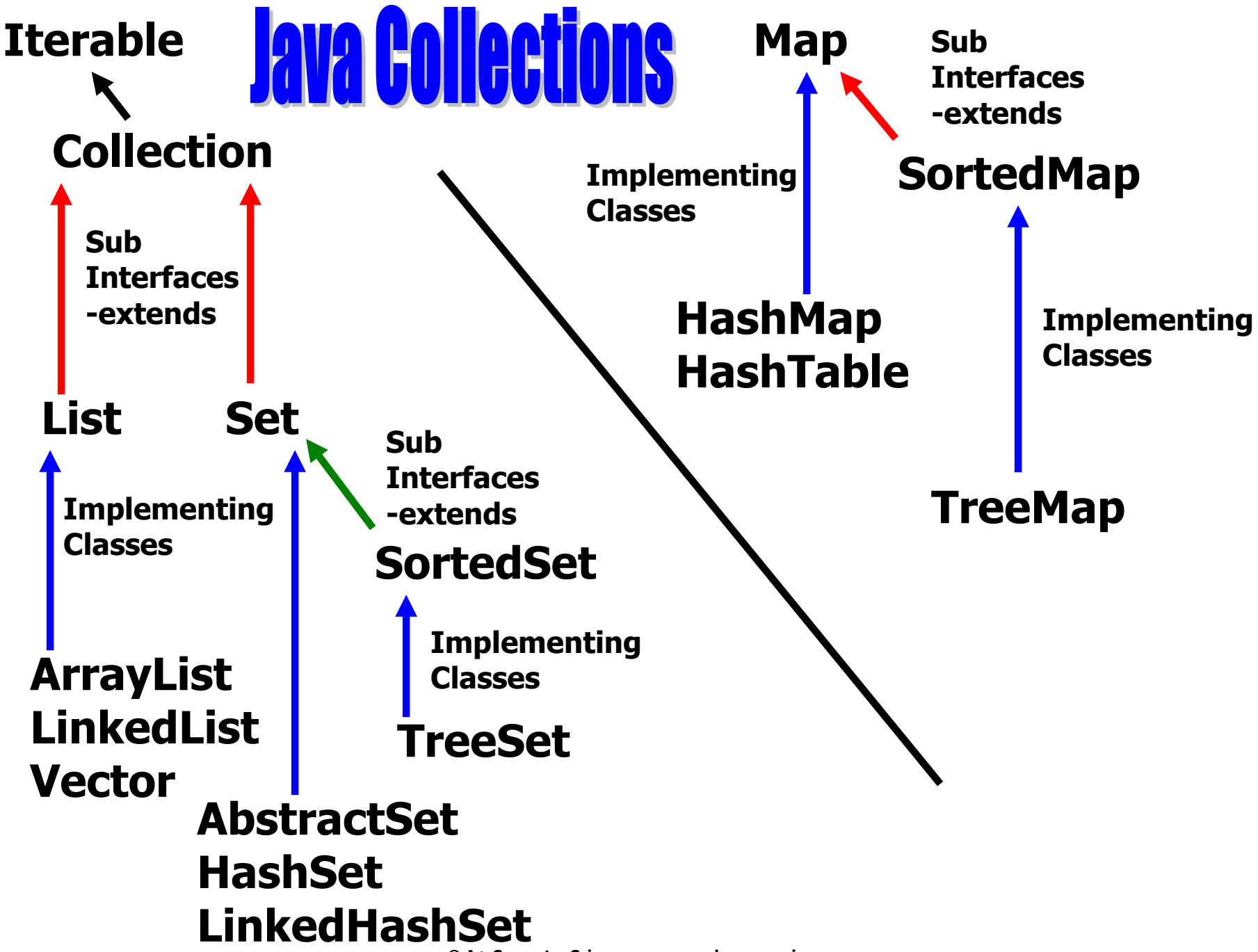
Maps

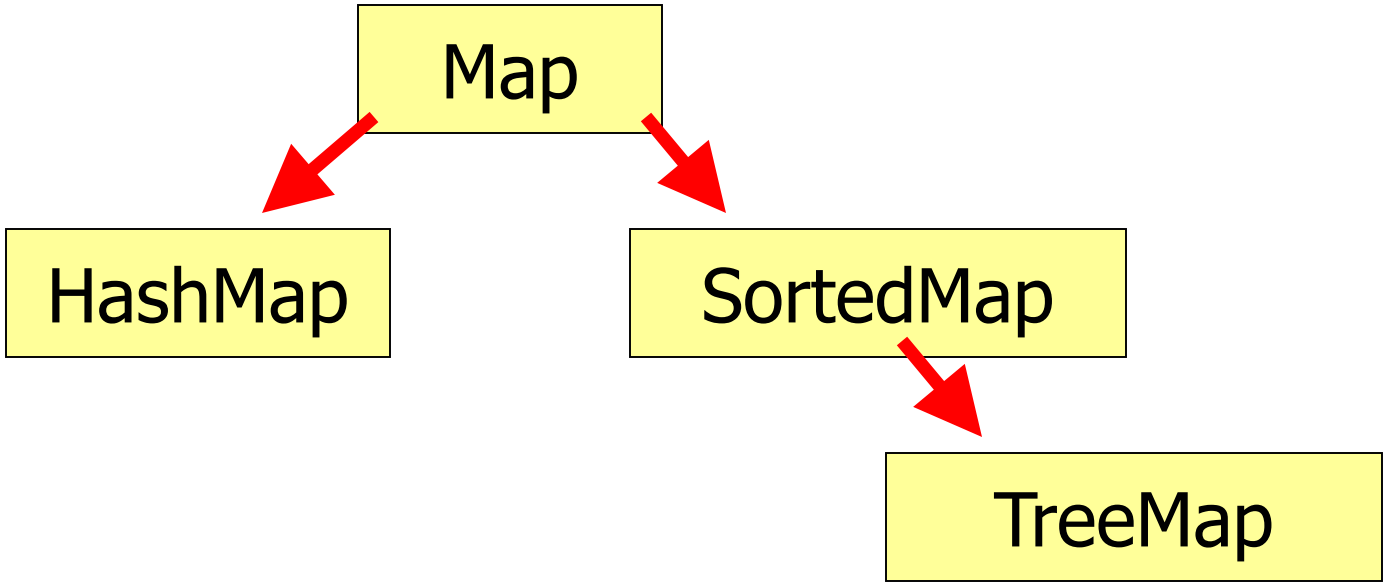


Lab 08



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.

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

MAPS

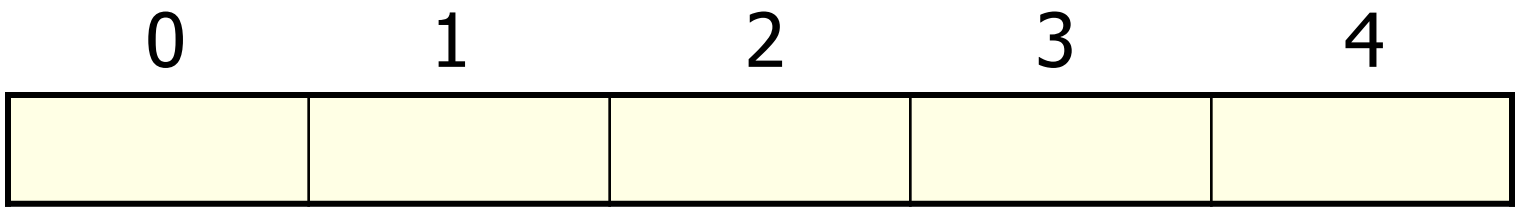
HashMap – a map ordered by each item's hashCode that is extremely time efficient.

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.

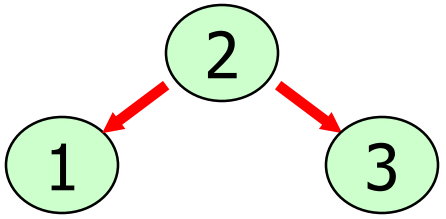
A hash table is a giant array. Each item is inserted into the array according to a hash formula.



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
values()	returns a collection view of the values contained in this 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
```

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);
System.out.println(map.put(1, 9.5));
System.out.println(map.put(2, 6.6));
System.out.println (map.put(5, 5.5));

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

OUTPUT

```
8.9
7.7
null

9.5
6.6
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, 1);
  else
    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'));
```

OUTPUT

2
null
4

c is not in the map,

so set count to 1

c is already in the map, so just bump up the count

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

open
treemapputtwo.java

map output

```
Map<Integer,Double> map;
map = new TreeMap<Integer,Double>();
map.put(4, 3.2);
map.put(1, 6.5);
map.put(2, 7.7);
System.out.println(map);
```

{1=6.5, 2=7.7, 4=3.2 }

map output

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

1 - 6.5
2 - 7.7
4 - 3.2

map output new

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

```
1 = 6.5
2 = 7.7
4 = 3.2
```

Open

treemapoutput.java

treemapoutputnew.java

open hashmapoutput.java

Big O

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 :

$O(1)$
$O(2^N)$
$O(N \log_2 N)$
$O(\log_2 N)$
$O(\log_2 N)$
$O(N^2)$
$O(N)$
$O(N^3)$

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 Lab 8