Lecture#8

Object Oriented Programming (JAVA)

Dr. Abu Nowshed Chy

Department of Computer Science and Engineering

University of Chittagong

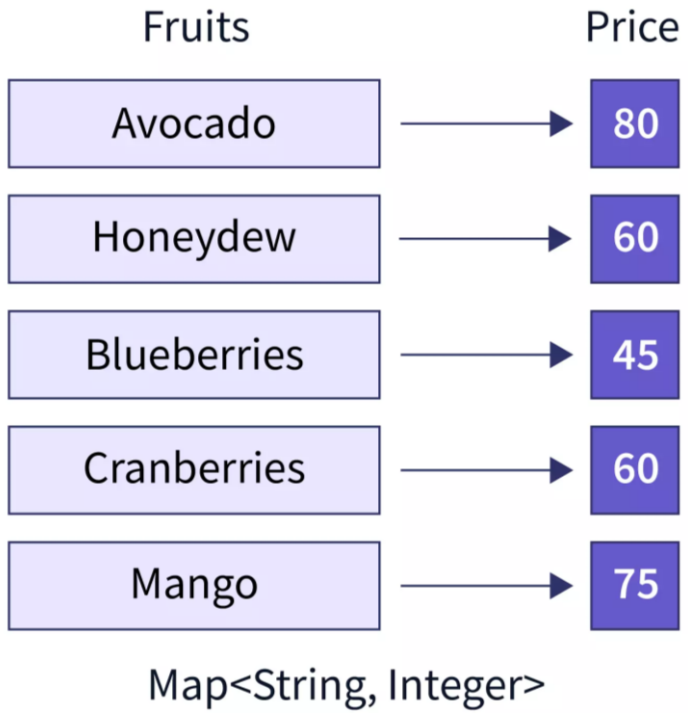
February 12, 2024

Faculty Profile

Map

- **Map**: Holds a set of unique *keys* and a collection of *values*, each associated with one value.
  - a.k.a. "dictionary", "associative array", "hash"

- Basic map operations:
  - **put**(*key*, *value*): Adds a mapping from a key to a value.
  - **get**(*key*): Retrieves the value mapped to the key.
  - **remove**(*key*): Removes the given key and its mapped value.



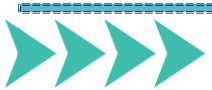
`myMap.get("Mango")` returns 75

Map

- Map is implemented by the HashMap and TreeMap classes
  - HashMap: implemented using an array called a "hash table"; extremely fast: O(1) ; keys are stored in unpredictable order
  - TreeMap: implemented as a linked "binary tree" structure; very fast: O(log N) ; keys are stored in sorted order

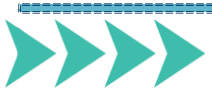
```
// maps from String keys to Integer values
Map<String, Integer> votes = new HashMap<String, Integer>();

// maps from String keys to Integer values
Map<String, Integer> votes = new TreeMap<String, Integer>();
```



Map

`put(`**key, value**`)`	adds a mapping from the given key to the given value; if the key already exists, replaces its value with the given one
`get(`**key**`)`	returns the value mapped to the given key (`null` if not found)
`containsKey(`**key**`)`	returns `true` if the map contains a mapping for the given key
`remove(`**key**`)`	removes any existing mapping for the given key
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value pairs in the map
`isEmpty()`	returns `true` if the map's size is 0
`toString()`	returns a string such as `"{a=90, d=60, c=70}"`
`keySet()`	returns a set of all keys in the map
`values()`	returns a collection of all values in the map
`putAll(`**map**`)`	adds all key/value pairs from the given map to this map
`equals(`**map**`)`	returns `true` if given map has the same mappings as this one



Map Example

```
package Similarity;

import java.util.HashMap;
import java.util.Map;

public class MapSampleCode {
    public static void main(String[] args) {
        Map<String,Integer> sampleMap=new HashMap<String,Integer>();
        sampleMap.put("Std1", 60);
        sampleMap.put("Std2", 70);
        sampleMap.put("Std3", 80);
        sampleMap.put("Std4", 90);

        System.out.println(sampleMap);

        System.out.println(sampleMap.get("Std1"));

        for(String mapKey:sampleMap.keySet()){
            System.out.println(mapKey);
        }
        for(String mapKey:sampleMap.keySet()){
            System.out.println(sampleMap.get(mapKey));
        }
    }
}
```

Map Practice Problem#1

Write a Java method that takes two binary string as input and estimate their simple matching coefficient (SMC) score as follows:

$\mathbf{x} = 1\,0\,0\,0\,0\,0\,0\,0\,0\,0$

$\mathbf{y} = 0\,0\,0\,0\,0\,0\,1\,0\,0\,1$

$f_{01} = 2$ (the number of attributes where p was 0 and q was 1)

$f_{10} = 1$ (the number of attributes where p was 1 and q was 0)

$f_{00} = 7$ (the number of attributes where p was 0 and q was 0)

$f_{11} = 0$ (the number of attributes where p was 1 and q was 1)

$$\text{SMC} = (f_{11} + f_{00}) / (f_{01} + f_{10} + f_{11} + f_{00})$$
$$= (0+7) / (2+1+0+7) = 0.7$$

Map Practice Problem#2

Consider the following two documents:

doc_1 = "Data is the new oil of the digital economy"

doc_2 = "Data is a new oil"

The Jaccard similarity of these two documents is estimated as follows:

$$J(doc_1, doc_2) = \frac{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cap \{'data', 'is', 'a', 'new', 'oil'\}}{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cup \{'data', 'is', 'a', 'new', 'oil'\}}$$

$$= \frac{\{'data', 'is', 'new', 'oil'\}}{\{'data', 'a', 'of', 'is', 'economy', 'the', 'new', 'digital', 'oil'\}}$$

$$= \frac{4}{9} = 0.444$$

Write a Java method that takes two string as input and return their Jaccard similarity Score.

List → ArrayList

It is an ordered collection of objects in which duplicate values can be stored. Since List preserves the insertion order, it allows positional access and insertion of elements.

```
// Creating a List
List<String> al = new ArrayList<>();

// Adding elements in the List
al.add("mango");
al.add("orange");
al.add("Grapes");

// Iterating the List
// element using for-each loop
for (String fruit : al)
    System.out.println(fruit);
```

Java

List Methods

Method	Description
add(int index, element)	This method is used with Java List Interface to add an element at a particular index in the list. When a single parameter is passed, it simply adds the element at the end of the list.
addAll(int index, Collection collection)	This method is used with List Interface in Java to add all the elements in the given collection to the list. When a single parameter is passed, it adds all the elements of the given collection at the end of the list.
clear()	This method is used to remove all the elements in the list.
contains(element)	Used to check whether a specific element is present in the List or not.
isEmpty()	This method is used to check whether the List is empty or not.
remove(element)	This method is used with Java List Interface to remove the first occurrence of the given element in the list.
removeAll(collection)	This method is used to remove all the elements from the collection which are present in the List.
size()	This method is used with Java List Interface to return the size of the list.

Array to ArrayList

```
private static final String colors[] = { "red", "white",
"blue", "green", "gray", "orange", "tan", "white",
"cyan", "peach", "gray", "orange" };

List mylist = new ArrayList( Arrays.asList( colors ) );
```

Java

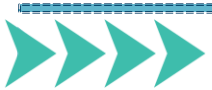
Set

It is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the mathematical set.

```
// Set demonstration using HashSet
Set<String> Set = new HashSet<String>();

// Adding Elements
Set.add("one");
Set.add("two");
Set.add("three");
Set.add("four");
Set.add("five");

// Set follows unordered way.
System.out.println(Set);
```



Set Methods

Method	Description
add(element)	This method is used to add a specific element to the set. The function adds the element only if the specified element is not already present in the Set.
addAll(collection)	This method is used to append all of the elements from the mentioned collection to the existing set.
clear()	Used to remove all the elements from the set but not delete the set.
contains(element)	Used to check whether a specific element is present in the Set or not.
isEmpty()	This method is used to check whether the set is empty or not.
remove(element)	This method is used to remove the given element from the set.
removeAll(collection)	This method is used to remove all the elements from the collection which are present in the set.
size()	This method is used to get the size of the set. This returns an integer value which signifies the number of elements.
toArray()	This method is used to form an array of the same elements as that of the Set.

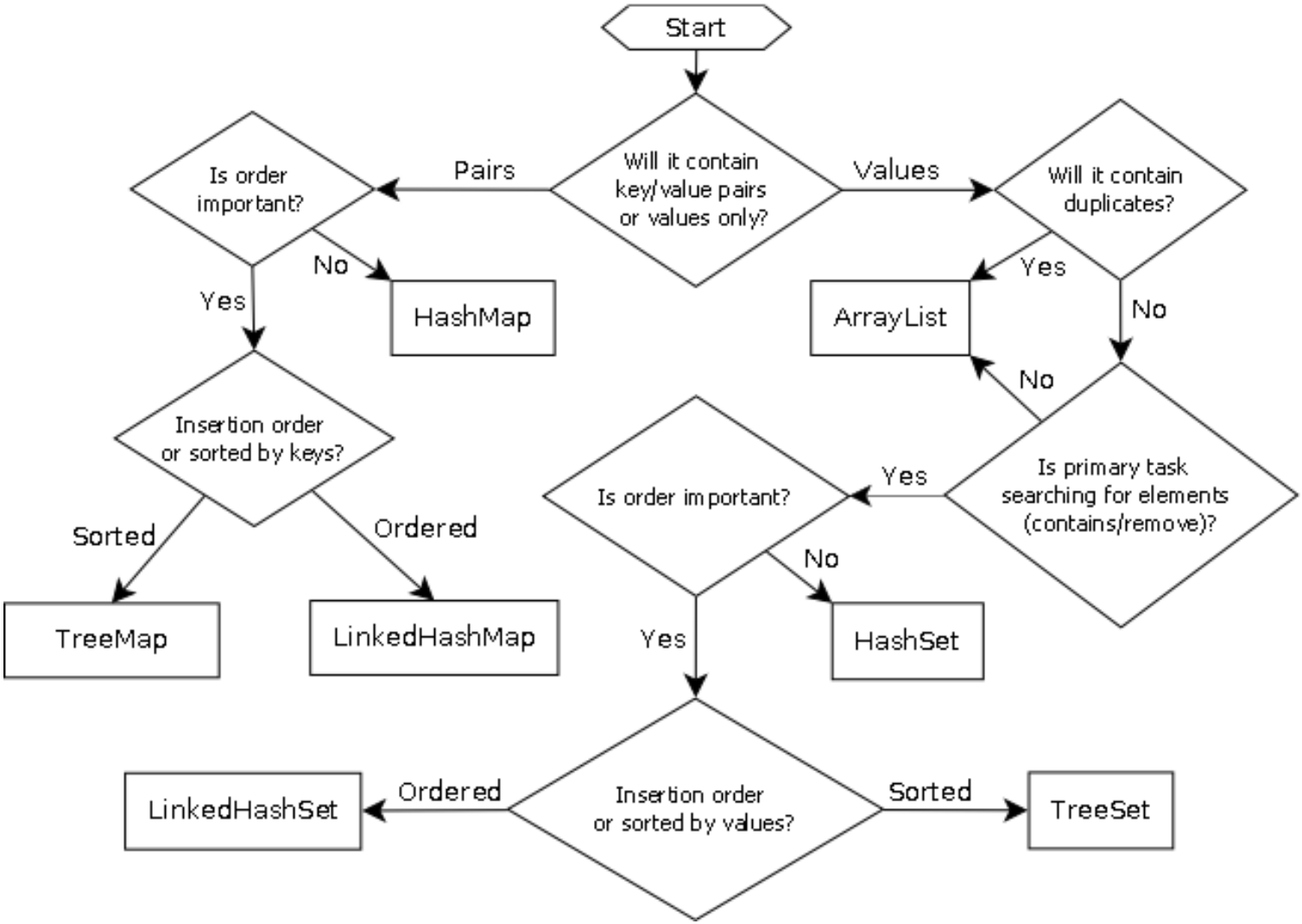
Difference between List, Set, and Map

List	Set	Map
The list interface allows duplicate elements	Set does not allow duplicate elements.	The map does not allow duplicate elements
The list maintains insertion order.	Set do not maintain any insertion order.	The map also does not maintain any insertion order.
We can add any number of null values.	But in set almost only one null value.	The map allows a single null key at most and any number of null values.
List implementation classes are Array List, LinkedList.	Set implementation classes are HashSet, LinkedHashSet, and TreeSet.	Map implementation classes are HashMap, HashTable, Tree Map, ConcurrentHashMap, and LinkedHashMap.
If you need to access the elements frequently by using the index then we can use the list	If you want to create a collection of unique elements then we can use set	If you want to store the data in the form of key/value pair then we can use the map.

Collection Summary

Collection	Ordering	Benefits	Weaknesses
array	by index	fast; simple	little functionality; cannot resize
`ArrayList`	by insertion, by index	random access; fast to modify at end	slow to modify in middle/front
`LinkedList`	by insertion, by index	fast to modify at both ends	poor random access
`TreeSet`	sorted order	sorted; O(log N)	must be comparable
`HashSet`	unpredictable	very fast; O(1)	unordered
`LinkedHashSet`	order of insertion	very fast; O(1)	uses extra memory
`TreeMap`	sorted order	sorted; O(log N)	must be comparable
`HashMap`	unpredictable	very fast; O(1)	unordered
`LinkedHashMap`	order of insertion	very fast; O(1)	uses extra memory
`PriorityQueue`	natural/comparable	fast ordered access	must be comparable

Choosing a Collection




Thank You!