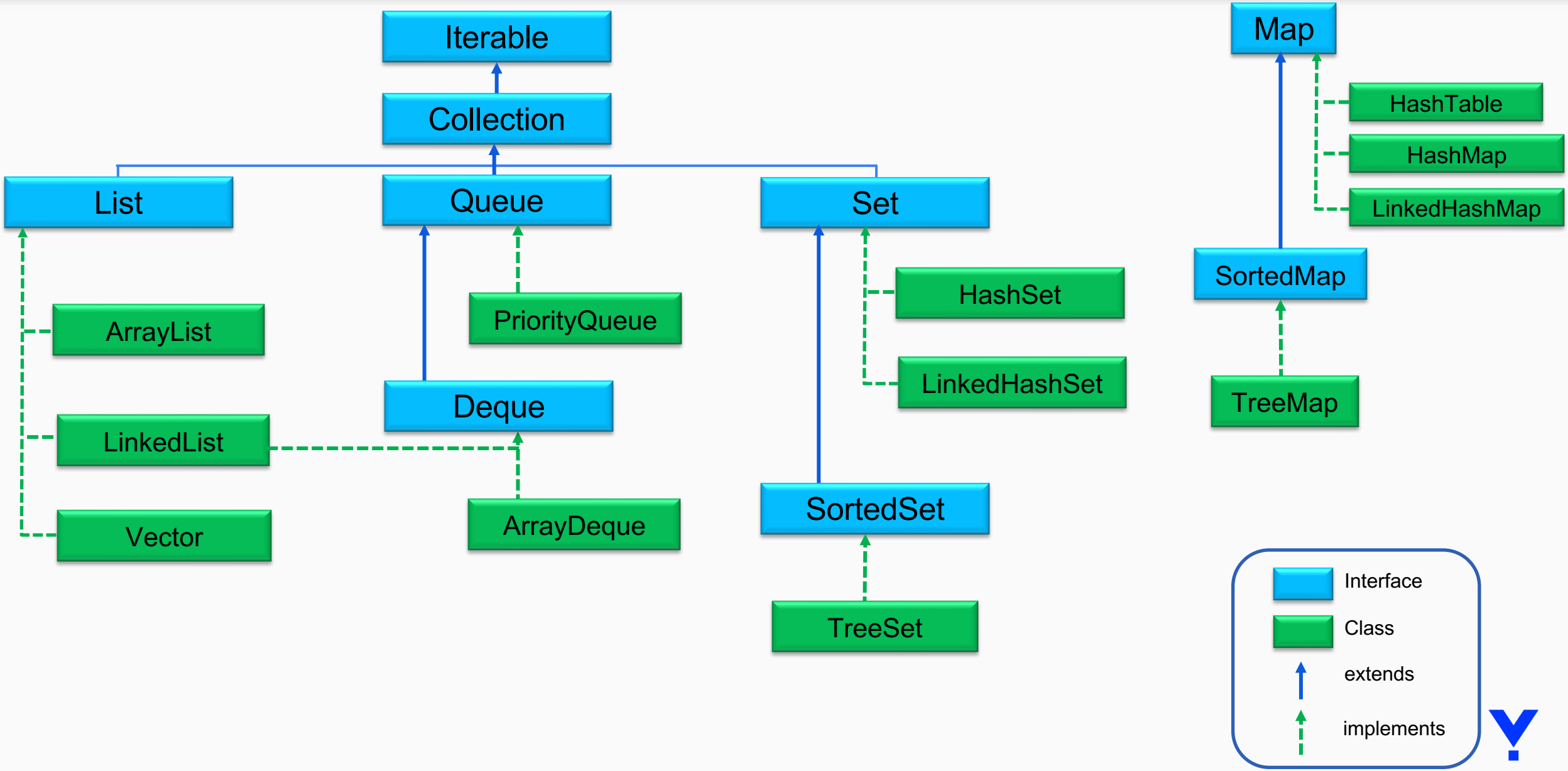
Data Structures and Algorithms Course

Collections Review

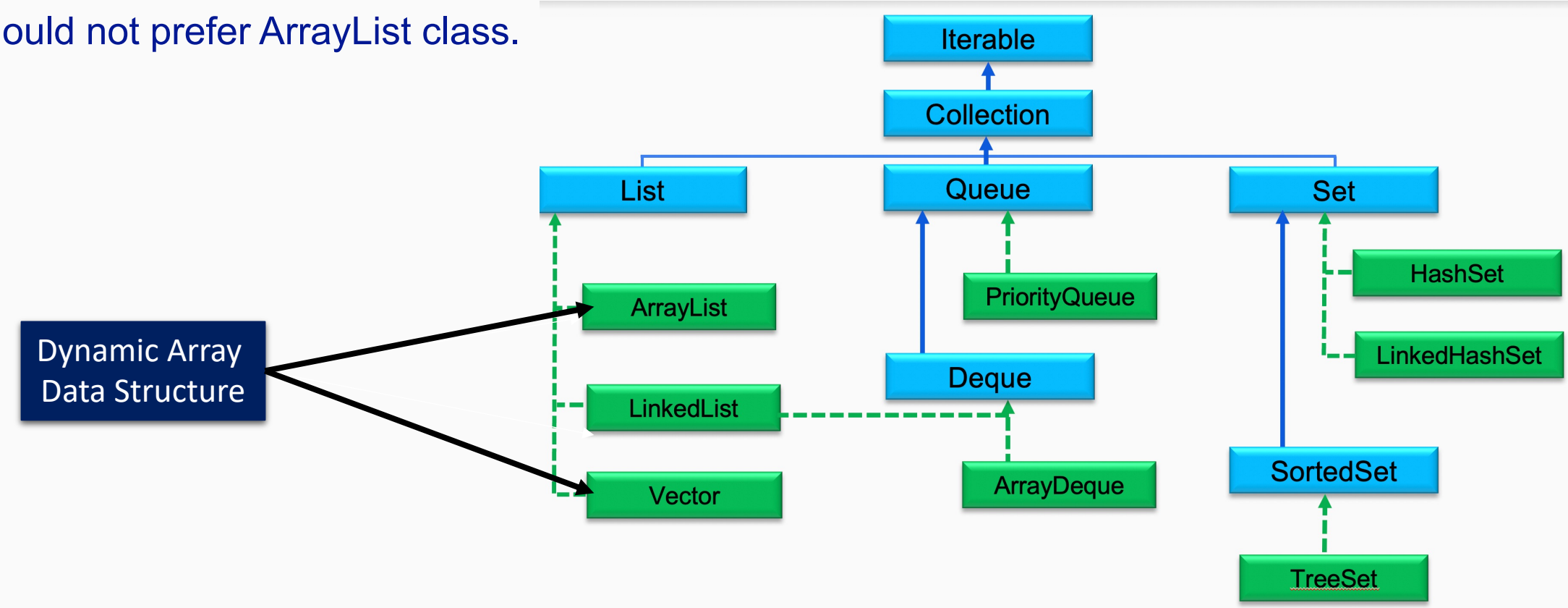
What is Collections Framework?



ArrayList

There two implementations of Dynamic Arrays in Java:

- **Vector class :** Size is increased by %100 if full. Synchronized (Only a single thread can access in multi-threaded environment)
- **ArrayList class:** Size is increased by % 50 if full. If you need multithreads access to data you should not prefer ArrayList class.



ArrayList- Basic skills for problem solving

- **How do we convert an Array to ArrayList?**
  - We can use Arrays.asList() method.

    Syntax : *<T> List<T> = asList(T... a)*

    Example:

    Integer[] arr={1,2,3};

    - *List<Integer> list = Arrays.asList(arr);*
    - *ArrayList<Integer> list= new ArrayList<>(Arrays.asList(1,2,3));*

  - We can use Collections.addAll().

    Syntax : *public static <T> boolean addAll(Collection<? super T> c, T... elements)*

    Example:

    Integer[] arr= new Integer[3];
    ArrayList<Integer> list= new ArrayList<>();
    Collections.addAll(list, arr);

ArrayList- Basic skills for problem solving

- **How do we convert an ArrayList to Array?**

  - We can use toArray() method.

    Syntax : *public Object[] toArray()*

    Example: *List<Integer> nums=new ArrayList<Integer>();*

    Integer[] arr=nums.toArray(new Integer[0]);

  - We can use streams.

    Example: *int[] arr = list.stream().mapToInt(i -> i).toArray();*

ArrayList- Basic skills for problem solving

- **How do we sort an ArrayList?**

  - We can sort an ArrayList in two ways ascending and descending order.
  - The Collections class provides two methods to sort an ArrayList in Java.

    1. Collections.sort() : Returns an ArrayList sorted in the ascending order

       Syntax: *Collections.sort(objectOfArrayList);*

       Example: *Collections.sort(list);*

    2. Collections.reverseOrder(): Sorts ArrayList in descending order

       Syntax: *Collections.sort(objectOfArrayList, Collections.reverseOrder());*

       Example: *Collections.sort(list, Collections.reverseOrder());*

Complexity???

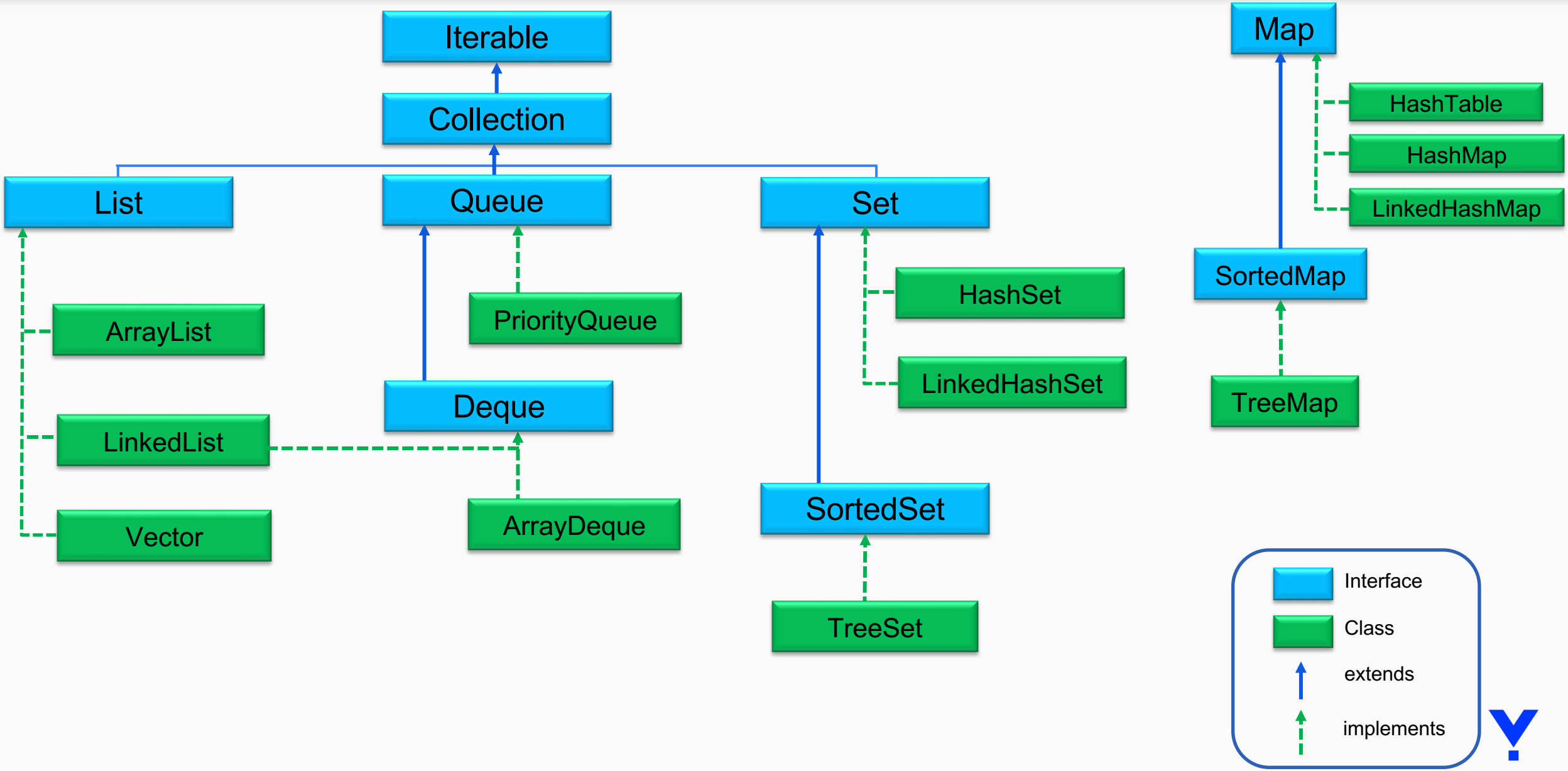
ArrayList- Basic skills for problem solving

- **How do we make a Binary search on an ArrayList?**
  - We can use binarySearch() method.
  - Searches the specified list for the specified object using the binary search algorithm. The list must be sorted into ascending order according to the natural ordering of its elements (as by the sort(List) method) prior to making this call. If it is not sorted, the results are undefined. If the list contains multiple elements equal to the specified object, there is no guarantee which one will be found.
  - This method runs in log(n) time for a "random access" list (which provides near-constant-time positional access). If the specified list does not implement the RandomAccess interface and is large, this method will do an iterator-based binary search that performs O(n) link traversals and O(log n) element comparisons.

    Syntax : *int binarySearch(listName, key);*

    Example: *Collections.binarySearch(nums, 2);*

What is Collections Framework?



HashMap- Basic skills for problem solving

- How to get values stored in the map?

  Map.values(): Returns a Collection view of the values contained in this map.

  Syntax : *Collection<V> values();*

  Example: *List<String> list=studentsMap.values();*

- How to get values stored in the map in a asc sorted order?

  List<String> list=map.values().stream().sorted((o1, o2)-> o1.compareToIgnoreCase(o2))

  .collect(Collectors.toList());

HashMap- Basic skills for problem solving

- How to get keys stored in the map?

  Map.keySet(): Returns a Set view of the keys contained in this map.

  Syntax : *Set<K> keySet();*

  Example: *Set<String> keySet = map.keySet();*

- How to get keys and values stored in the map?

  Map.entrySet() : A map entry (key-value pair). The Entry may be unmodifiable, or the value may be modifiable if the optional setValue method is implemented.

  Syntax : *Set<Map.Entry<K, V>> entrySet();*

  Example: *Set<Map.Entry<Integer,String>> entrySet= Map.entrySet();*

Differences between HashMap and HashTable:

- HashMap is a non-synchronized data structure. It is not thread-safe and cannot be shared across many threads without the use of synchronization code, while Hashtable is synchronized. It's thread-safe and can be used by several threads.
- HashMap supports one null key and numerous null values, whereas Hashtable does not.
- If thread synchronization is not required, HashMap is often preferable over HashTable.

Why does HashMap allow null whereas HashTable does not allow null?

- The objects used as keys must implement the hashCode and equals methods in order to successfully save and retrieve objects from a HashTable.
- Hashcode and Equals cannot be implemented by null because it is not an object.
- HashMap is a more advanced and improved variant of Hashtable.
- HashMap was invented after HashTable to overcome the shortcomings of HashTable.

Differences between HashMap and TreeMap

HashMap	TreeMap
The Java HashMap implementation of the Map interface is based on hash tables.	Java TreeMap is a Map interface implementation based on a Tree structure.
The Map, Cloneable, and Serializable interfaces are implemented by HashMap.	NavigableMap, Cloneable, and Serializable interfaces are implemented by TreeMap.
Because HashMap does not order on keys, it allows for heterogeneous elements.	Because of the sorting, TreeMap allows homogenous values to be used as a key.
HashMap is quicker than TreeMap because it offers O(1) constant-time performance for basic operations such as to get() and put().	TreeMap is slower than HashMap because it performs most operations with O(log(n)) performance, such as add(), remove(), and contains().
A single null key and numerous null values are allowed in HashMap.	TreeMap does not allow null keys, however multiple null values are allowed.
To compare keys, it uses the Object class's equals() method. It is overridden by the Map class's equals() function.	It compares keys using the compareTo() method.
HashMap does not keep track of any sort of order.	The elements are arranged in chronological sequence (ascending).
When we don't need a sorted key-value pair, we should use the HashMap.	When we need a key-value pair in sorted (ascending) order, we should use the TreeMap.

Algo Problem: Letter Candles

Your friend Alice has a box with N letter candles in it. The cost of the box is determined as follows - Find the number of occurrences of each characters in the box and sum up the squares of these numbers.

Alice wants to reduce the cost of the box by removing some candles from it. However, she is allowed to remove at most M candles from the box. Can you help Alice determine the minimum cost of the box?

Input

The first line of the input contains the integer N, representing the number of letter candles. The second line of the input contains the integer M, representing the number of candles Alice can

remove.

The third line of the input contains an N-lettered string S, which contains lowercase English letters, representing the letter candles in the box.

Algo Problem: Letter Candles

Output

Print the minimum possible cost of the box.

Example #1

Input:

6

2

bacacc

Output:

6

Explanation: There are two As, one B, and three Cs in the box. Current cost of the box is $2^2 + 1^2 + 3^2 = 14$. The best way to minimize the cost of the box is to remove two C-shaped candles from it. The new minimal cost will be $2^2 + 1^2 + 1^2 = 6$. The answer is 6.

Algo Problem: Letter Candles

Example #2

Input:

15

3

XXXXXXXXXXXXXXX

Output:

144

Explanation: There are 15 Xs. The current cost of the box is $15^2 = 225$. The only way to minimize the cost is by reducing three X-shaped candles from it. The new minimal cost will be $12^2 = 144$. The answer is 144

Algo Problem: Total weight of same items

You are given two 2D integer arrays, items1 and items2, representing two sets of items. Each array items has the following properties:

- items[i] = [$value_i$, $weight_i$] where $value_i$ represents the **value** and $weight_i$ represents the **weight** of the i^{th} item.
- The value of each item in items is **unique**.

Return *a 2D integer array* return *where* return[i] = [$value_i$, $weight_i$]*, with* $weight_i$ *being the* ***sum of weights*** *of all items with value* $value_i$.

Note: return should be returned in **ascending** order by value.

Algo Problem: Total weight of same items

Example 1:
Input: items1 = [[1,1],[4,5],[3,8]], items2 = [[3,1],[1,5]]

Output: [[1,6],[3,9],[4,5]]

Explanation:

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 5, total weight = 1 + 5 = 6.

The item with value = 3 occurs in items1 with weight = 8 and in items2 with weight = 1, total weight = 8 + 1 = 9.

The item with value = 4 occurs in items1 with weight = 5, total weight = 5.

Therefore, we return [[1,6],[3,9],[4,5]].

Algo Problem: Total weight of same items

Example 2:
Input: items1 = [[1,1],[3,2],[2,3]], items2 = [[2,1],[3,2],[1,3]]

Output: [[1,4],[2,4],[3,4]]

Explanation:

The item with value = 1 occurs in items1 with weight = 1 and in items2 with weight = 3, total weight = 1 + 3 = 4.

The item with value = 2 occurs in items1 with weight = 3 and in items2 with weight = 1, total weight = 3 + 1 = 4.

The item with value = 3 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4.

Therefore, we return [[1,4],[2,4],[3,4]].

Algo Problem: Total weight of same items

Example 3:

Input: items1 = [[1,3],[2,2]], items2 = [[7,1],[2,2],[1,4]]

Output: [[1,7],[2,4],[7,1]]

Explanation:

The item with value = 1 occurs in items1 with weight = 3 and in items2 with weight = 4, total weight = 3 + 4 = 7.

The item with value = 2 occurs in items1 with weight = 2 and in items2 with weight = 2, total weight = 2 + 2 = 4.

The item with value = 7 occurs in items2 with weight = 1, total weight = 1. Therefore, we return

[[1,7],[2,4],[7,1]].