Learning Consolidation

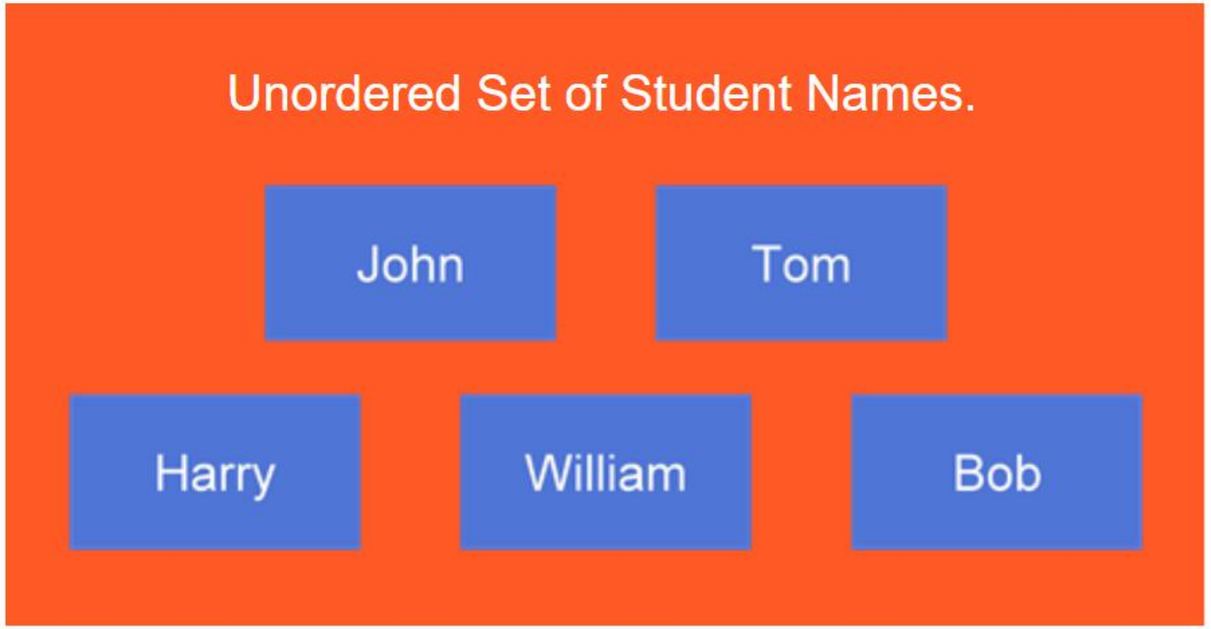
Manipulate Objects Using Unordered Collections and Construct Objects as a Key Value Pair

Learning Objectives

- Describing Set Interface
- Implementing HashSet and TreeSet
- Knowing the Difference Between List and Set
- Understanding Map Interface
- Implementing HashMap and TreeMap
- Iterating Through Map

Set Interface

- Set is an interface that is part of the collection framework.
- It is an **unordered** collection of objects where duplicate values cannot be stored.
- Set has methods to add, remove, clear, and size to enhance the usage of this interface.
- There is no indexing on the elements of the set.
- Hence, a for loop cannot be used on a set, but the Iterator can be used.



TreeSet Implementation

```
TreeSet<Integer> oddNumbers = new TreeSet<>();
oddNumbers.add(23);
oddNumbers.add(7);
oddNumbers.add(3);
oddNumbers.add(19);
oddNumbers.add(21);
oddNumbers.add(21);
System.out.println(oddNumbers);
System.out.println("First element: "+ oddNumbers.first());
System.out.println("Last element: "+ oddNumbers.last());
System.out.println("PollFirst:" + oddNumbers.pollFirst());
System.out.println("PollLast: " + oddNumbers.pollLast());
System.out.println(oddNumbers);
```

```
[3, 7, 19, 21, 23]
First element: 3
Last element: 23
PollFirst:3
PollLast: 23
[7, 19, 21]
```

The TreeSet contains only sorted elements:

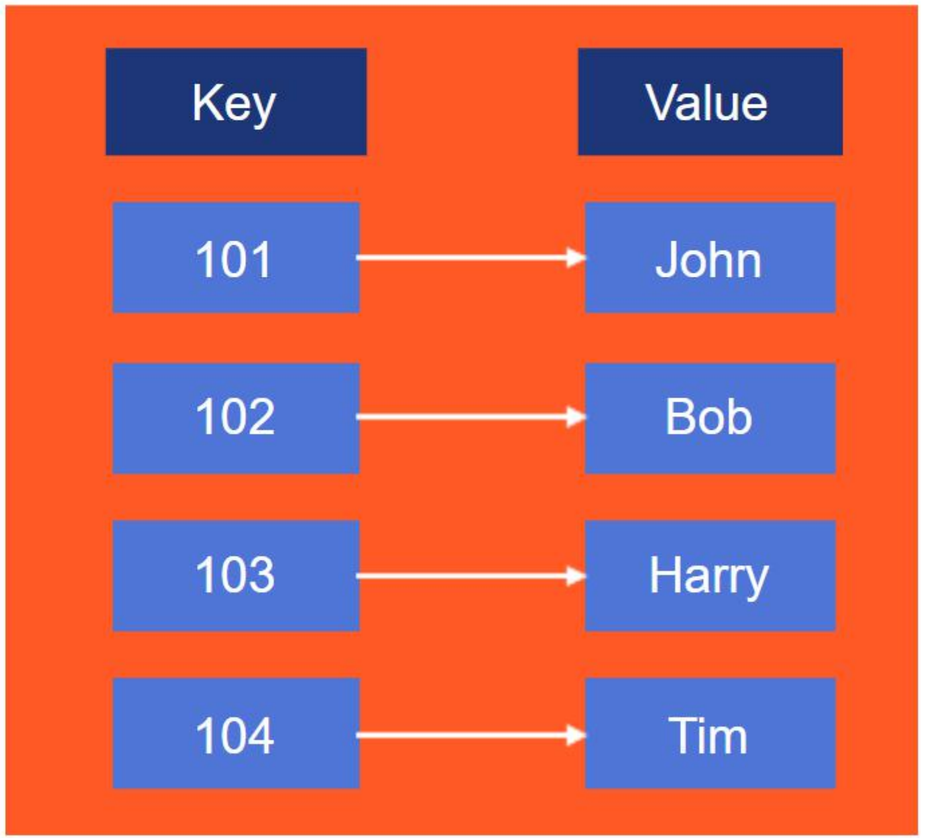
- first()- Returns the first (lowest) element currently in this set.
- last() - Returns the last (highest) element currently in this set.
- pollFirst - Retrieves and removes the first (lowest) element or returns null if this set is empty.
- pollLast - Retrieves and removes the last (highest) element or returns null if this set is empty.

List vs. Set

List	Set
A list is an ordered sequence of objects.	A set is an unordered sequence of objects.
It allows duplicate elements.	The set is unique and does not allow duplicate elements.
As the list has indexing, elements are accessed by their position.	There is no indexing in the set. So, elements cannot be accessed by their position.
A list can hold multiple null elements.	A set can only store one null element.

Map Interface

- The Map interface enables you to create a collection with key-value pair objects.
- The Map interface is not a subtype of the Collection interface.
- A map containing the key as an integer and the value as a string is shown below. The key is always unique.



```java
Map<Integer,String> hashMap = new HashMap<>();
hashMap.put( k: 101, v: "John");
hashMap.put( k: 102, v: "Johny");
hashMap.put( k: 103, v: "Bob");
System.out.println(hashMap);
System.out.println("Does Map contains key 102 :" +hashMap.containsKey( o: 102));
System.out.println("Does Map contains value Bob :" +hashMap.containsValue( o: "Bob"));
```

```
{101=John, 102=Johny, 103=Bob}
Does Map contains key 102 :true
Does Map contains value Bob :true
```

HashMap Implementation

- HashMap having a key of type integer and value of type string is declared.
- Since the map does not extend the collection, the add method is not available to a map.

Methods of the Map Interface:

- Put (key,value) - Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced with a new value.
- ContainsKey (Object key) - Returns true if this map contains a mapping for the specified key.
- containsValue(Object value)-Returns true if this map maps one or more keys to the specified value.

Use Iterators to Iterate Through the Map Object

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

// enter name/url pair
map.put( k: "101",  v: "Java");
map.put( k: "102",  v: "JavaScript");
map.put( k: "104",  v: "Angular");
map.put( k: "105",  v: "React");

// using iterators
Iterator<Map.Entry<String, String>> itr = map.entrySet().iterator();

while(itr.hasNext())
{
    Map.Entry<String, String> entry = itr.next();
    System.out.println("Key = " + entry.getKey() +
            ", Value = " + entry.getValue());
}
```