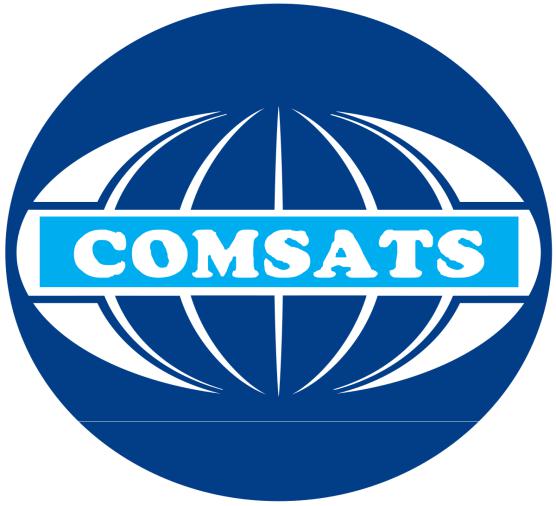
**Object Oriented Programming Lab**

****

**Assignment 3rd**

**Submitted to: Sir-Muhammad Ali-Khan**

**Submitted by: Nasratullah (FA18-BCS-401)**

**Submission Date: 6th June ,2021.**

**Section: B**

**Question:**

**Take an example of your own choice and apply the following:**

**1. Map**

**2. Set**

**3. Linked List**

**Map Interface:**

* Stores Key/value pairs
* A Map is an object that maps key to the value.
* A map cannot contain duplicate keys (Key are unique).

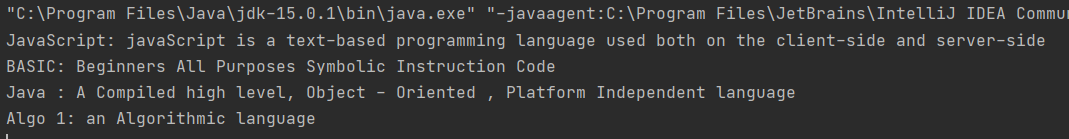
1. A Single key only appears one in the Map
2. A key can map to only one value

* Values do not have to be unique.

**Example:**

|  |
| --- |
| package LABAssig3;  import java.util.HashMap; import java.util.Map;   public class MapProgram {   public static void main(String[] args) {  Map<String,String> languages= new HashMap();   languages.put("Java :","A Compiled high level, Object - Oriented , Platform Independent language");  languages.put("Algo 1:","an Algorithmic language");  languages.put("BASIC:","Beginners All Purposes Symbolic Instruction Code");  languages.put("JavaScript:","javaScript is a text-based programming language used both on the client-side and server-side");   for( String Key : languages.keySet()) {  System.*out*.println(Key + " " + languages.get(Key));  }  } } |

**Example:**



**Map Method:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| Object put(object key, Object value | It is used to insert an entry in this map. |
| Void putAll(Map map) | It is used to insert the specified map in this map |
| Object remove (Object key) | It is to delete an entry for the specified key. |
| Object get (Object key) | It is used to return the value for the specified key |
| Boolean containsKey(Object key) | It is used to search the specified key from this map. |
| Set keyset() | It is used to return the Set view containing all The keys. |
| Set entrySet () | It is used to return the Set view containing all the keys and values. |

**Set Interface:**

* A Set is Unordered and has no duplicates.
* Defines two fundamental methods.

1. Boolean add (Object o) – reject duplicates
2. Iterator iterator ()

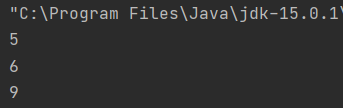
* Provides an Iterator to step through the elements in the set.

1. No guaranteed order in the basic Set interface
2. There is a Sorted Set interface that extends Set.

**Example:**

|  |
| --- |
| package LABAssig3;  import java.util.HashSet; import java.util.Set;  public class SetProgram {   public static void main(String[] args) {  Set<Integer> values = new HashSet<>();   values.add(5);  values.add(6);  values.add(9);    for (int i : values){  System.*out*.println(i);  }  } } |

**Output:**

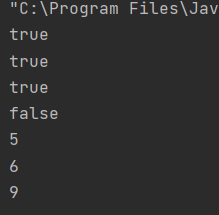
****

**Set Does not support Duplicate Elements:**

**Example:**

|  |
| --- |
| package LABAssig3;  import java.util.HashSet; import java.util.Set;  public class SetProgram {   public static void main(String[] args) {  Set<Integer> values = new HashSet<>();   System.*out*.println(values.add(5));  System.*out*.println(values.add(6));  System.*out*.println(values.add(9));  System.*out*.println(values.add(6));    for (int i : values){  System.*out*.println(i);  }  } } |

**Output:**

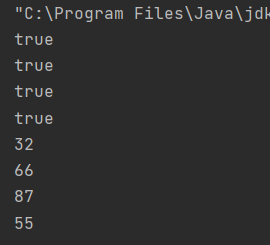
****

If We see the output which is unordered Hash Set Give value which is not in the Sequence.

**Ex:**

|  |
| --- |
| package LABAssig3;  import java.util.HashSet; import java.util.Set;  public class SetProgram {   public static void main(String[] args) {  Set<Integer> values = new HashSet<>();   System.*out*.println(values.add(87));  System.*out*.println(values.add(32));  System.*out*.println(values.add(55));  System.*out*.println(values.add(66));   for (int i : values){  System.*out*.println(i);  }  } } |

**Output:**

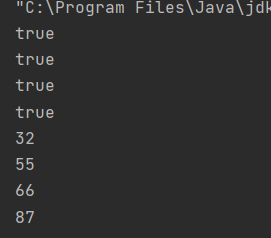
****

If you want to get value in sequence or order (Ascending order) then you have to use Tree set<>.

**Ex:**

|  |
| --- |
| package LABAssig3;  import java.util.HashSet; import java.util.Set; import java.util.TreeSet;  public class SetProgram {   public static void main(String[] args) {  Set<Integer> values = new TreeSet<>();   System.*out*.println(values.add(87));  System.*out*.println(values.add(32));  System.*out*.println(values.add(55));  System.*out*.println(values.add(66));   for (int i : values){  System.*out*.println(i);  }  } } |

**Output:**

****

**Linked List Class:**

* Stores each element in a node
* Each node stores a link to the next and previous nodes
* Insertion and removal are inexpensive

1. Just update the links in the surrounding nodes

* Linear traversal is inexpensive
* Random access is expensive

1. Start from beginning or end and traverse each node while counting.

**Linked List methods:**

* The list is sequential, so access it that way

a)

* ListIterator knows about position

1. Use add() from ListItrator to add at a position
2. Use remove() from ListIterator to remove at a position

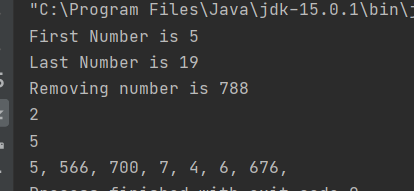
* Linked List know a few things too

1. Void addFirst (Object o), void addLast(Object o)
2. Object getFirst(), Object getLast()
3. Object removeFirst(), Object removeLast()

**Example:**

|  |
| --- |
| package LABAssig3;  import java.util.LinkedList;  public class LinkedListEx {   public static void main(String[] args) {  LinkedList<Integer> Ll1 = new LinkedList<>();  LinkedList<Integer> Ll2 = new LinkedList<>();  Ll2.add(15);  Ll2.add(18);  Ll2.add(19);   Ll1.add(6);  Ll1.add(7);  Ll1.add(4);  Ll1.add(6);  Ll1.add(0, 5);  Ll1.add(1, 15);  Ll1.addLast(676);  System.*out*.println("First Number is "+Ll1.getFirst());  System.*out*.println("Last Number is "+Ll2.getLast());  Ll1.addFirst(788);  System.*out*.println("Removing number is "+Ll1.removeFirst());  System.*out*.println(Ll1.indexOf(6));  System.*out*.println(Ll1.lastIndexOf(6));  //l1.clear();  Ll1.set(1, 566);  Ll1.set(2, 700);  //Ll2.set(3,500);  for(int i=0; i<Ll1.size(); i++){  System.*out*.print(Ll1.get(i));  System.*out*.print(", ");  }  } } |

**Output:**



**List VS Set:**

* List can contain duplicate elements whereas Set contains unique elements only.