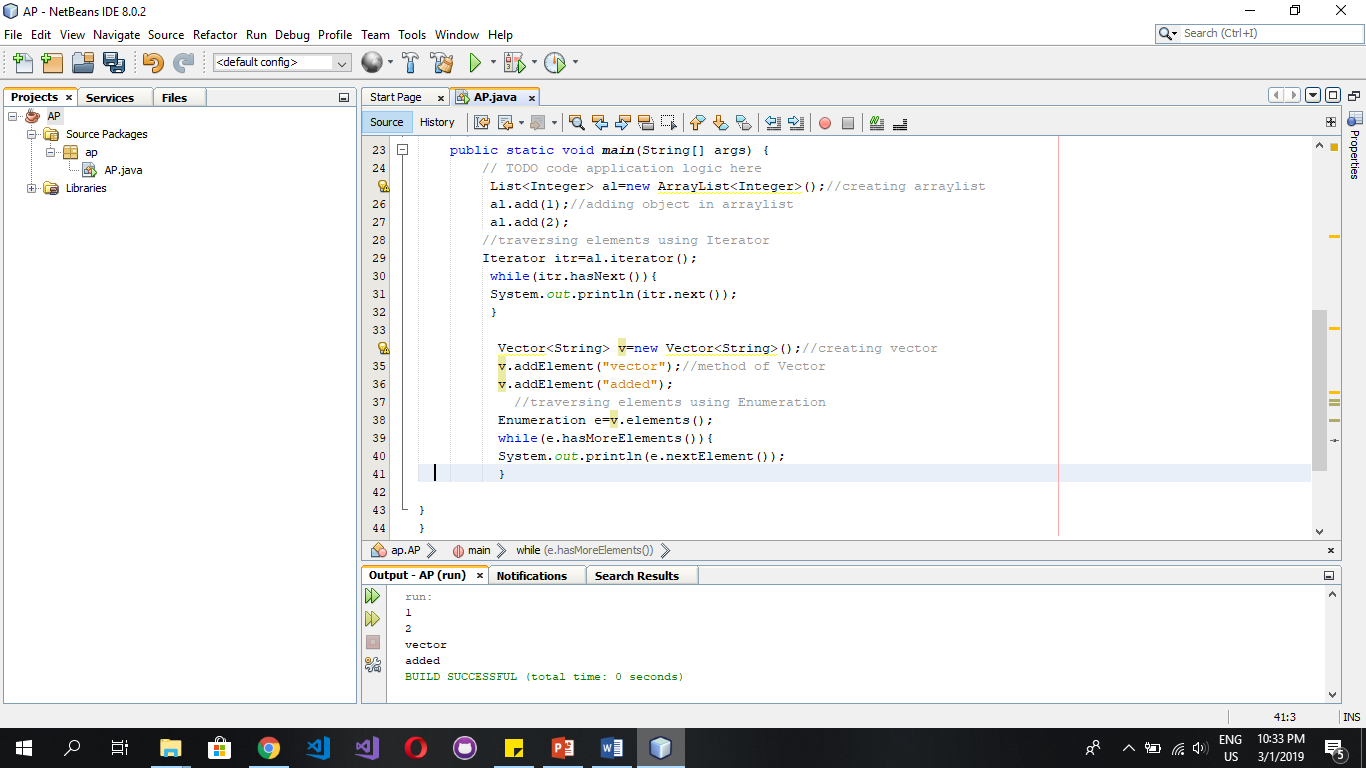
Advance Programming(Java)

Difference between ArrayList and Vector:

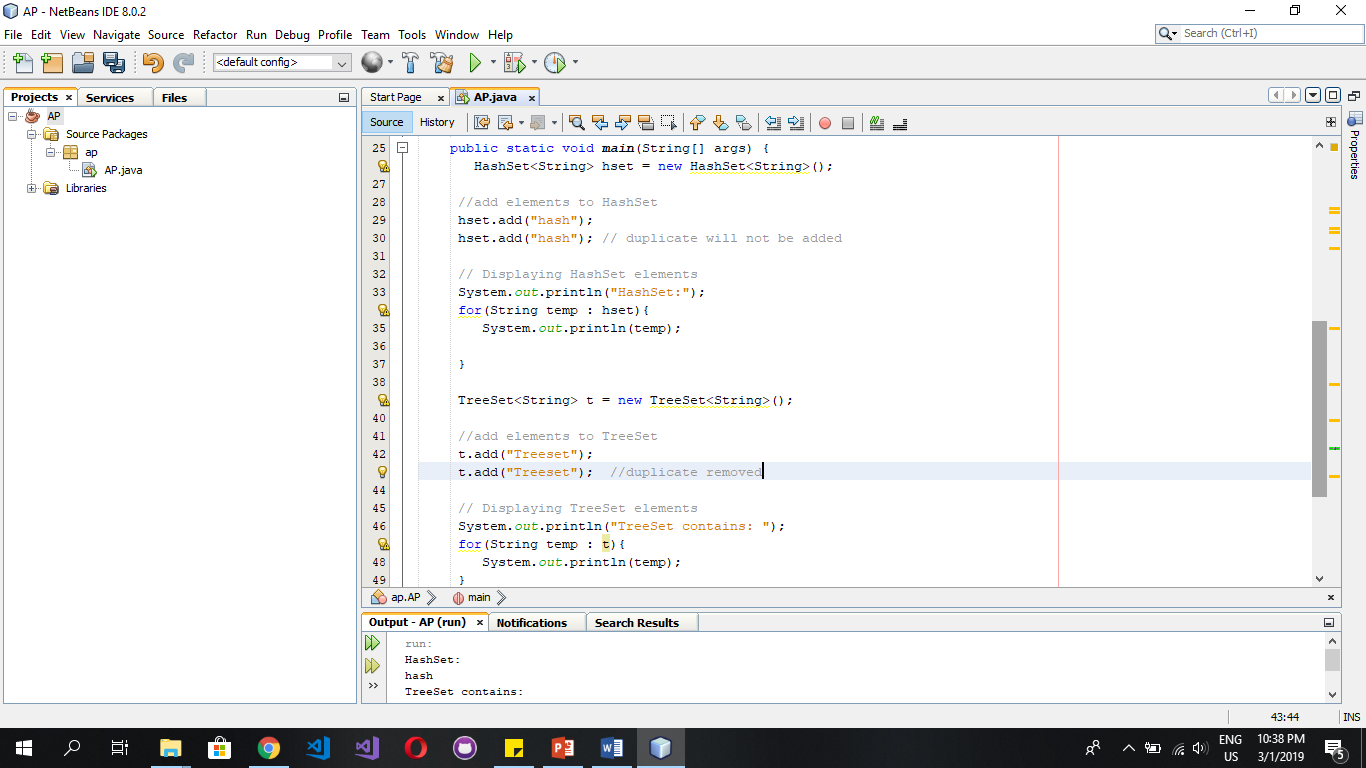
|  |  |  |
| --- | --- | --- |
|  | ArrayList | Vector |
| Syntax: | ArrayList<T> a = new ArrayList<T>();  Where T can be string or any other class. | Vector<T> v = new Vector<T>();  Where T can be string or any other class. |
| Traversal: | ArrayList can only be traversed by using  Iterator. | Vector can be traversed by using iterator as well as enumeration. |
| Performance: | Performance of arraylist operations is good as compared to vector. | Vector operations performance is slow as compare to arraylist. |
| Growing capacity: | ArrayList can grow 50% of its size. | Vector can grow 100% of its size. |
| Synchronization: | Arraylist is non-synchronized. | Vector is syschronized. |
| Multi-threading: | Multiple Threads can work on arraylist. | Vector does not support multi-threading. |

Code snippet:



Difference between HashSet and TreeSet:

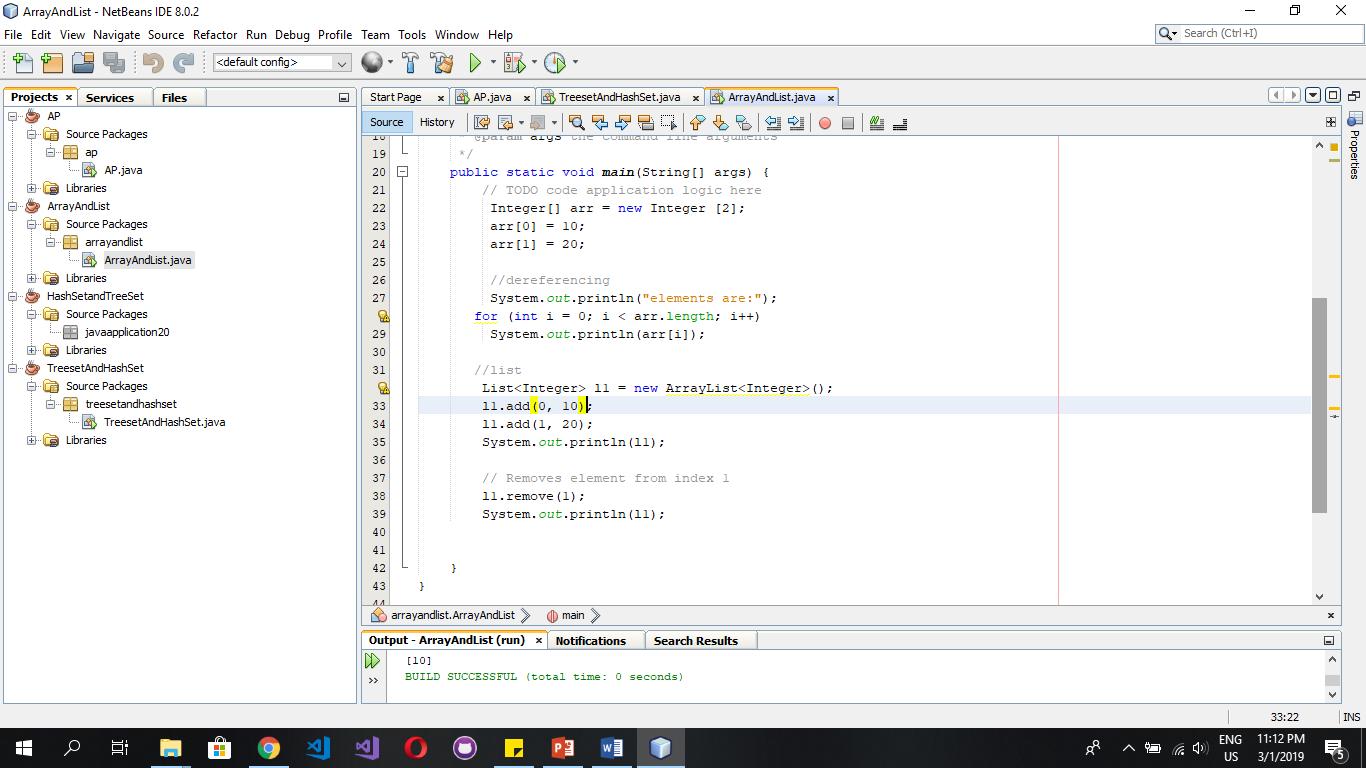
|  |  |  |
| --- | --- | --- |
|  | HashSet | TreeSet |
| Syntax: | HashSet<String> h = new HashSet<String>(); | TreeSet<String> a=new TreeSet<String>(); |
| Performance: | Performance of HashSet operations is good as compared to vector. | TreeSet operations performance is slow as compare to arraylist. |
| Time Complexity: | Searching, deletion or updation can be done in constant time. | It takes O(logn) for search, delete and update. |
| Ordering: | It does not maintain order. | It maintains order. |
| Objects with null value: | It does not throw exception in case of null objects. | It throws an exception in case of null objects. |



Code snippet:

Difference between Array and List:

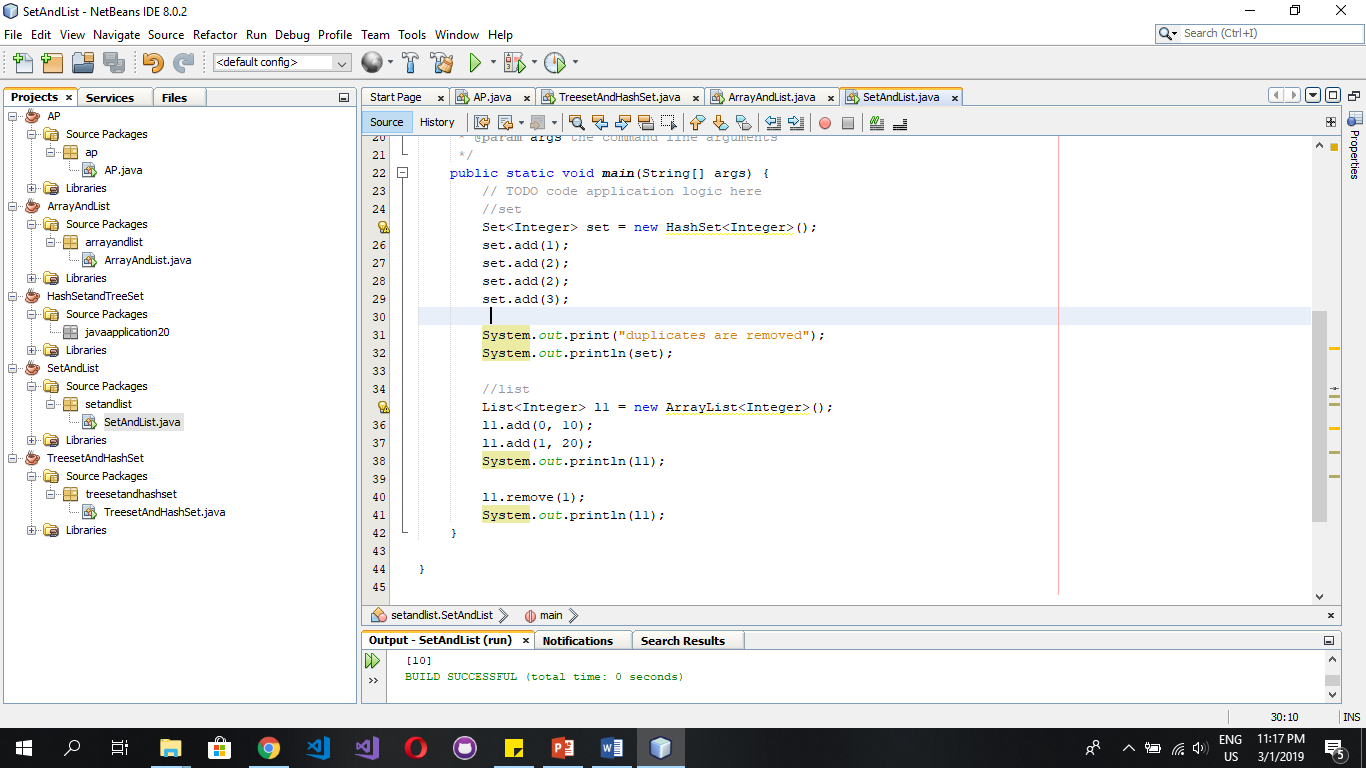
|  |  |  |
| --- | --- | --- |
|  | Array | List |
| Syntax: | Int array = new int[10]; | List <T> list = new List<T>(); |
| Fixed Length: | Array is a fixed length data structure. | List is dynamic in size. |
| Access modifier: | Array members are accessed using dereference operator i.e [] | List uses its method to access its members. |
| Parent Interface: | Array implements the interfaces cloneable and java.io.serializable. | List has parent interface called collection. |

Code snippet:

Difference between Set and List:

|  |  |  |
| --- | --- | --- |
|  | Set | List |
| Syntax: | Set<Integer> Set = new HashSet< Integer >(); | List <T> list = new List<T>(); |
| Ordered: | Set is an unordered collection. | List is an ordered collection. |
| Duplicate: | Set does not allow duplicates. | List allow duplicates. |
| Implementations: | Implementations of set are Hashset, linked hashset, treeset etc. | Implementations of list are arraylist,  Linkedlist etc. |
| Traversal: | Set can be travesed through iterator. | Listiterator is used to traverse elements of a list. |

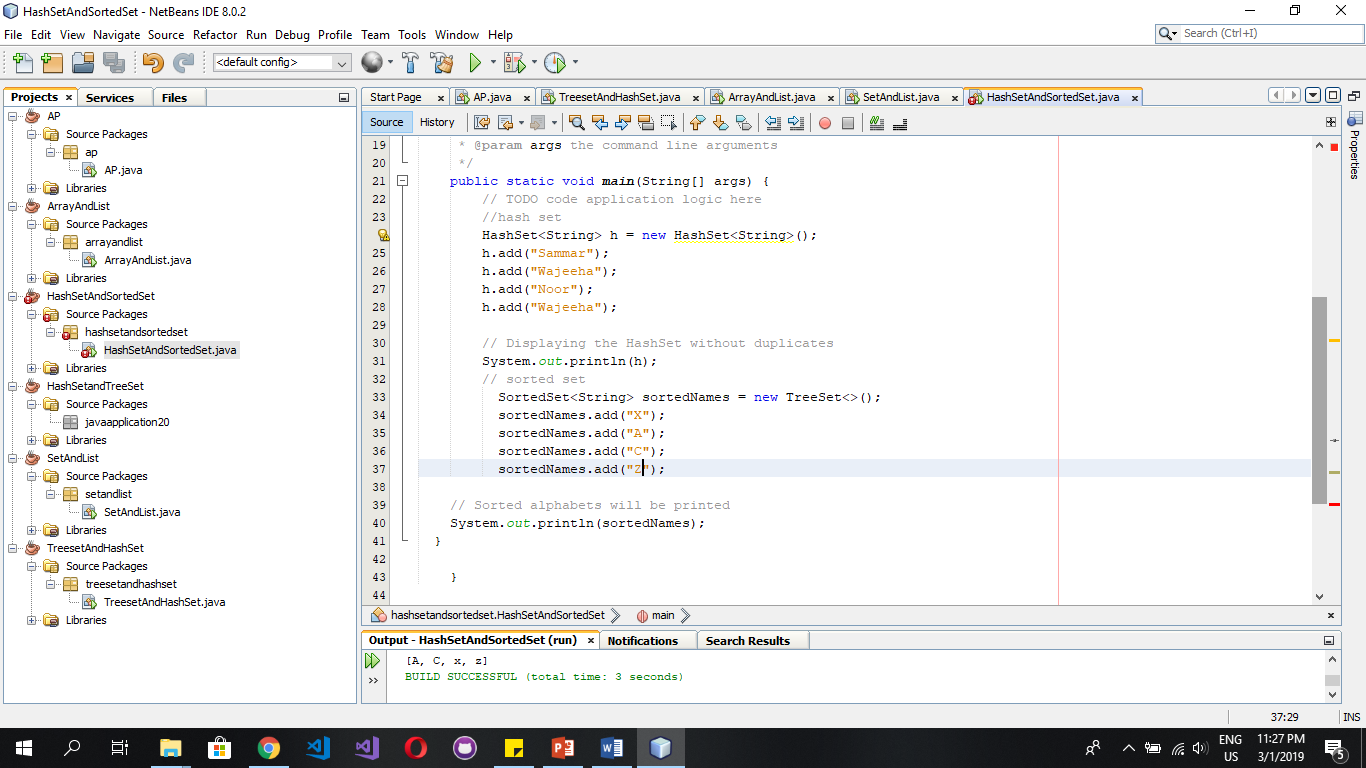
Code Snippet:-



Difference between HashSet and SortedSet:

|  |  |  |
| --- | --- | --- |
|  | HashSet | SortedSet |
| Syntax: | HashSet h = new HashSet(); | SortedSet s = new SortedSet(); |
| Interface: | Implements Set Interface. | SortedSet is an interface in collection framework. |
| Underlying Data structure: | Hash table. | List is an ordered collection. |
| Ordered: | Order is not guaranteed. | Order is guaranteed. |
| Null elements: | Null elements are allowed in hashset. | Sorted set doesnot allow null objects and they throw exception in case of null objects. |
| Time complexity: | O(1). | O(N). |
| Searching time: | Searching an element can be done in constant time. | Searching an element in sorted set can be done in O(N). |

Code snippet:



Difference between NavigableSet and NavigableMap:

|  |  |  |
| --- | --- | --- |
|  | NavigableSet | NavigableMap |
| Syntax: | NavigableSet<String> Set = new TreeSet<String>(); | NavigableMap<Integer,Integer> Map=new TreeMap<Integer, Integer>(); |
| Inheritance: | Extends sorted set interface. | Extends sorted map. |
| Representation: | Represent a sorted set. | Represent a sorted map. |
| Key association: | There is no key in navigable sets. | Key value is associated with inserted element so that element can be searched through key. |