Results:

Our three implementation is generally use same complexity if you can control Load factor and using a well hash code method to make no collision.

We calculate in the Time complexity rapor each data structure has different approching to reach the positon of value. To best usage of these method is having a good hash code before the comparision of these implementation.

For large arrays using Coalesed implementation is good because we are keeping local next, if it keep some other values we keep next2 and prev using these datas we can reach the as fastest as possible but this is cost much space If we have a space problem we can use (after coalesed techice) is Hash Tree Set.

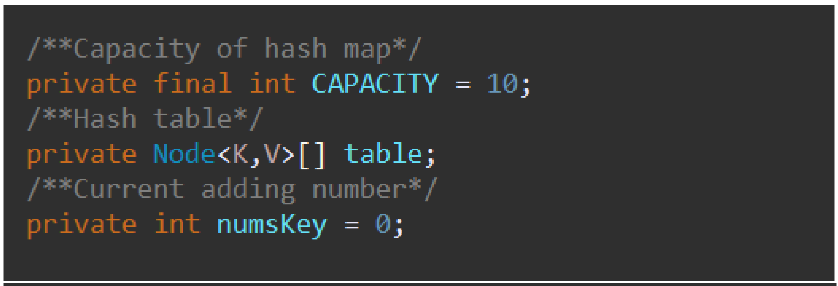
For medium arrays using Hash Tree Set will be the best choice because It dont want space as coalesed techice and Tree set reaching an element is faster than Linked List.

The second best is option for medium array if you need fast you can use coalesed techice but if you dont have enough memory you can use Hash Linked List

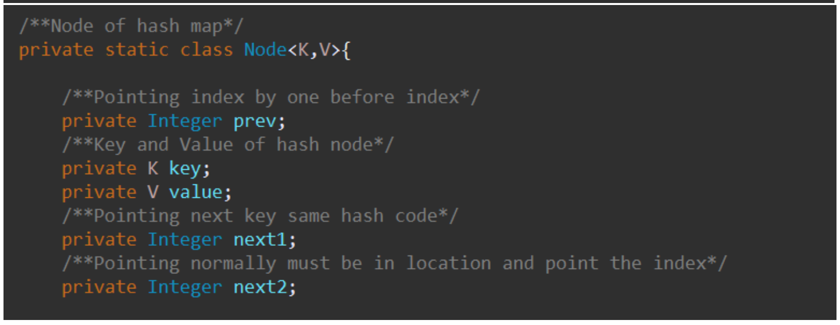
For Small arrays using Hash Linked List will be the best choice because.For small array we can assume the linear is constant for small array in reach, remove or add method.

And also it need less space according to the others.The second best option would be the Hash Tree Set. Hash Tree Set has same opportinitues about space in linked list according to the Coalesed Techice

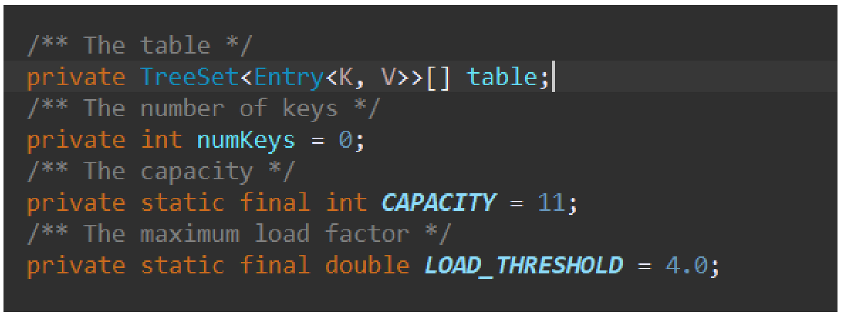
Nodes Of Implementation:



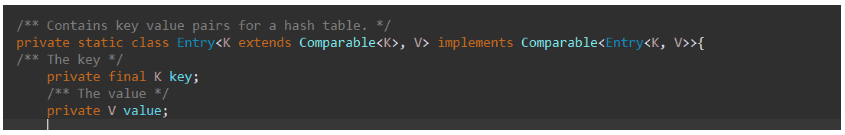
```java
/**Capacity of hash map*/
private final int CAPACITY = 10;
/**Hash table*/
private Node<K,V>[] table;
/**Current adding number*/
private int numsKey = 0;
```



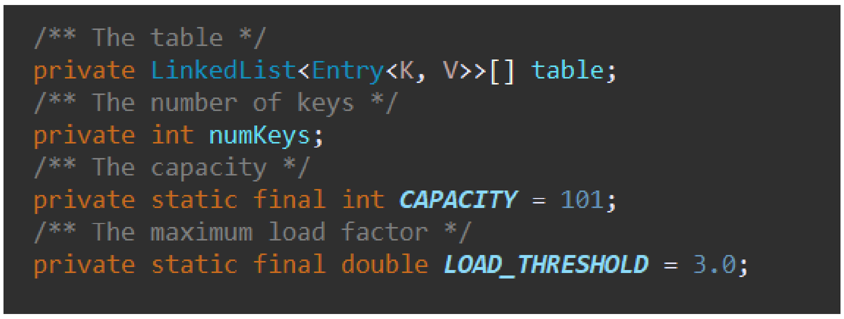
```java
/**Node of hash map*/
private static class Node<K,V>{

    /**Pointing index by one before index*/
    private Integer prev;
    /**Key and Value of hash node*/
    private K key;
    private V value;
    /**Pointing next key same hash code*/
    private Integer next1;
    /**Pointing normally must be in location and point the index*/
    private Integer next2;
```



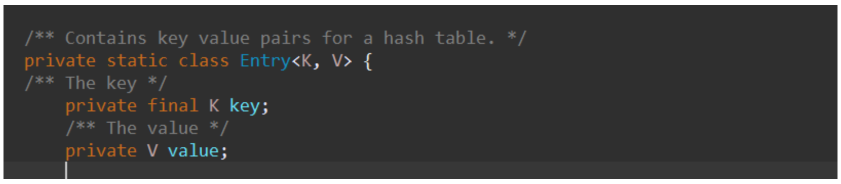
```java
/** The table */
private TreeSet<Entry<K, V>>[] table;
/** The number of keys */
private int numKeys = 0;
/** The capacity */
private static final int CAPACITY = 11;
/** The maximum load factor */
private static final double LOAD_THRESHOLD = 4.0;
```



```java
/** Contains key value pairs for a hash table. */
private static class Entry<K extends Comparable<K>, V> implements Comparable<Entry<K, V>>{
/** The key */
    private final K key;
    /** The value */
    private V value;
```



```java
/** The table */
private LinkedList<Entry<K, V>>[] table;
/** The number of keys */
private int numKeys;
/** The capacity */
private static final int CAPACITY = 101;
/** The maximum load factor */
private static final double LOAD_THRESHOLD = 3.0;
```



```java
/** Contains key value pairs for a hash table. */
private static class Entry<K, V> {
/** The key */
    private final K key;
    /** The value */
    private V value;
```