References used for creating this document:
https://www.geeksforgeeks.org/

https://stackoverflow.com/

https://www.javatpoint.com/

```
import java.io.*;
import java.util.*;

class ArrayListExample {
    public static void main(String[] args)
    {
        // Size of the
        // ArrayList
        int n = 5;

        // Declaring the ArrayList with
        // initial size n
        ArrayList<Integer> arrli
            = new ArrayList<Integer>(n);

        // Appending new elements at
        // the end of the list
        for (int i = 1; i <= n; i++)
            arrli.add(i);

        // Printing elements
        System.out.println(arrli);

        // Remove element at index 3
        arrli.remove(3);

        // Displaying the ArrayList
        // after deletion
        System.out.println(arrli);

        // Printing elements one by one
        for (int i = 0; i < arrli.size(); i++)
            System.out.print(arrli.get(i) + " ");
    }
}
```

```
ArrayList<String> cars = new ArrayList<String>(); // Create an
ArrayList object
```

```
cars.set(0, "Opel");
```

```
cars.remove(0);
```

To remove all the elements in the `ArrayList`, use the `clear()` method:

Example

```
cars.clear();
```

ArrayList Size

To find out how many elements an ArrayList have, use the `size` method:

Example

```
cars.size();

Collections.sort(cars);  // Sort cars
```

TREESET

TreeSet in Java

TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided. This must be consistent with equals if it is to correctly implement the Set interface. It can also be ordered by a Comparator provided at set creation time, depending on which constructor is used. The TreeSet implements a NavigableSet interface by inheriting AbstractSet class.

```
// Java program to demonstrate TreeSet
import java.util.*;

class TreeSetExample {

    public static void main(String[] args)
    {
        TreeSet<String> ts1 = new TreeSet<String>();

        // Elements are added using add() method
        ts1.add("A");
        ts1.add("B");
        ts1.add("C");

        // Duplicates will not get insert
        ts1.add("C");

        // Elements get stored in default natural
        // Sorting Order(Ascending)
        System.out.println(ts1);
```

```
    }
}


// Java code to demonstrate
// the working of TreeSet

import java.util.*;
class TreeSetDemo {

    public static void main(String[] args)
    {
        TreeSet<String> ts
            = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("Geek");
        ts.add("For");
        ts.add("Geeks");

        System.out.println("Tree Set is " + ts);

        String check = "Geeks";

        // Check if the above string exists in
        // the treeset or not
        System.out.println("Contains " + check
                           + " " + ts.contains(check));

        // Print the first element in
        // the TreeSet
        System.out.println("First Value " + ts.first());

        // Print the last element in
        // the TreeSet
        System.out.println("Last Value " + ts.last());

        String val = "Geek";

        // Find the values just greater
        // and smaller than the above string
        System.out.println("Higher " + ts.higher(val));
        System.out.println("Lower " + ts.lower(val));
    }
}
```

HashMap and HashSet both are one of the most important classes of Java Collection framework.

Following are the important differences between HashMap and HashSet.

Sr. No.	Key	HashMap	HashSet
1	Implementation	Hashmap is the implementation of Map interface.	Hashset on other hand is the implementation of set interface.
2	Internal implementation	Hashmap internally do not implements hashset or any set for its implementation.	Hashset internally uses Hashmap for its implementation.
3	Storage of elements	HashMap Stores elements in form of key-value pair i.e each element has its corresponding key which is required for its retrieval during iteration.	HashSet stores only objects no such key value pairs maintained.
4	Method to add element	Put method of hash map is used to add element in hashmap.	On other hand add method of hashset is used to add element in hashset.
5	Index performance	Hashmap due to its unique key is faster in retrieval of element during its iteration.	HashSet is completely based on object so compared to hashmap is slower.
6	Null Allowed	Single null key and any number of null value can be inserted in hashmap without any restriction.	On other hand Hashset allows only one null value in its collection,after which no null value is allowed to be added.

HASHSET EXAMPLE :

```
1. import java.util.*;
2. class HashSet1{
3.  public static void main(String args[]){
4.   //Creating HashSet and adding elements
5.     HashSet<String> set=new HashSet();
6.           set.add("One");
7.           set.add("Two");
8.           set.add("Three");
9.           set.add("Four");
10.          set.add("Five");
11.          Iterator<String> i=set.iterator();
12.          while(i.hasNext())
13.          {
14.          System.out.println(i.next());
15.          }
16. }
17.}
```

```
1. In this example, we see that HashSet doesn't allow duplicate elements.
2. import java.util.*;
3. class HashSet2{
4.  public static void main(String args[]){
5.   //Creating HashSet and adding elements
6.   HashSet<String> set=new HashSet<String>();
7.   set.add("Ravi");
8.   set.add("Vijay");
9.   set.add("Ravi");
10.  set.add("Ajay");
11.  //Traversing elements
12.  Iterator<String> itr=set.iterator();
13.  while(itr.hasNext()){
14.   System.out.println(itr.next());
15.  }
16. }
17.}
```

```
1. import java.util.*;
2. class HashSet3{
3.  public static void main(String args[]){
4.   HashSet<String> set=new HashSet<String>();
5.           set.add("Ravi");
6.           set.add("Vijay");
7.           set.add("Arun");
8.           set.add("Sumit");
9.           System.out.println("An initial list of elements: "+set);
10.          //Removing specific element from HashSet
```

```
        set.remove("Ravi");
        System.out.println("After invoking remove(object) method: "+set);
        HashSet<String> set1=new HashSet<String>();
        set1.add("Ajay");
        set1.add("Gaurav");
        set.addAll(set1);
        System.out.println("Updated List: "+set);
        //Removing all the new elements from HashSet
        set.removeAll(set1);
        System.out.println("After invoking removeAll() method: "+set);
        //Removing elements on the basis of specified condition
        set.removeIf(str->str.contains("Vijay"));
        System.out.println("After invoking removeIf() method: "+set);
        //Removing all the elements available in the set
        set.clear();
        System.out.println("After invoking clear() method: "+set);
 }
}
```

All classes extend `Object` class which has the `hashcode()` method.It usually returns a distinct value as it converts the internal address of an Object,but it is not necessarily required.

HASHMAP:

When we add a duplicate element with the same key and same value, then the key-value pair does not store second time.

```
import java.util.*;
public class HashMapExample
{
public static void main(String args[])
{
//creating object of HashMap
HashMap<String, Integer> hm= new HashMap<String, Integer>();
//adding key-value pair
hm.put("John", 23);
hm.put("Monty", 27 );
hm.put("Richard", 21);
hm.put("Devid", 19);
System.out.println("Before adding duplicate keys: ");
System.out.println(hm);
//adding duplicate keys
hm.put("Monty", 25);    //replace the Monty's previous age
hm.put("Devid", 19);
System.out.println("After adding duplicate keys: ");
System.out.println(hm);
}
}
```

```
// Java program to traversal a
// Java.util.HashMap

import java.util.HashMap;
import java.util.Map;

public class TraversalTheHashMap {
    public static void main(String[] args)
    {
        // initialize a HashMap
        HashMap<String, Integer> map = new HashMap<>();

        // Add elements using put method
        map.put("vishal", 10);
        map.put("sachin", 30);
        map.put("vaibhav", 20);

        // Iterate the map using
        // for-each loop
        for (Map.Entry<String, Integer> e : map.entrySet())
            System.out.println("Key: " + e.getKey()
```

```
                                + " Value: " + e.getValue());
    }
}
```

Ways to traverse through the Map: Examples:

Way1:

```
// using keySet() for iteration over keys
        for (String name : gfg.keySet())
            System.out.println("key: " + name);

        // using values() for iteration over values
        for (String url : gfg.values())
            System.out.println("value: " + url);
```

Way 2

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

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

```
        }
```

Way 3

```
// forEach(action) method to iterate map

        gfg.forEach((k,v) -> System.out.println("Key = "

                + k + ", Value = " + v));
```

Other methods:

```
// remove element with a key
         // using remove method
         hm.remove(4);

// Change Value using put method
         hm.put(1, "Geeks");
```