Collections of java as follows:

13/08/24

ArrayList: An Array list is resizable array implementation.

```
import Java.util.*;
class Array List ex{
   Public static void main (String[] args){
   ArrayList <string> List = new Array list <>( );
     List.add ("Apple");
     List.add ("Banana");
     List.add ("cherry");
    System.out.Println (List);
 }
}
```

output: [Apple, Banana, cherry]

Linked List:
A Linked list is a doubly linked list implementation of list Interface.

Program:

```
   import java.util.*;
   class Linked list ex {
    Public static void main (string args[]){
     Linked List < String > List = new Linked List <>();
       List.add ("Apple");
      List.add ("Cherry");
} }
```

output = [Apple, cherry]

Hash set = A Hash set is a set implementation that uses a hash table for storage.

Code =

```
import. Java.util.*;
Class Hash set ex {
      Public static void main (string args[]) {
        Hash set <string> set = new Hash set <>();
          Set.add ("Apple");
          Set.add ("Ice Cream");
          System.out.Println (set);
} }
```

output = [Apple, Ice Cream]

Tree Set = A Tree set is a set Implementation that uses a tree for storage.

Code =

```
   import Java.util.*;
  Class  Tree set ex {
        Public static void main (String Args[]) {
          Tree set <String> set = new Tree set <>();
            Set.add ("Apple");
        Set.add("Banana");
        Set.add("cherry");
       System.out.Println(set);
} }
```

output = [Apple, Banana, cherry]

HashMap = a Map Implementation that uses a hash table fo
storage.

```
mport Java.util.*;
Class HashMap ex{
    Public static void main (string args[]){
    Hashmap < String Integer > map = new HashMap <>();
     map.Put ("Apple", 1);
     map.Put ("Banana", 2);
     map.Put ("cherry", 3);
 system.out.Println (map);
}
```

Put = {Apple = 1, Banana = 2, cherry = 3}

ee Map = A 'TreeMap' is a map implementation that uses a
tree for storage.

```
e = import Java.util.*;
 class Tree map ex {
```

6) Priority Queue. A Priority Queue is a Queue Implementation that orders Elements Based on their natural ordering or a Custom Comparator.

Code:

```
 Import Java. util .*;
  class Priority Queue ex {
    Public static Void main (String [] args){
   Priority Queue < String> Queue = new Priority Queue <>();
        queue.add ("Apple");
        queue. add ("Banana");
        queue. add ("cherry");
   System. out. Printh (queue);
 }
} }
```

output- [Apple, Banana, cherry].

Array Dequeue.

An Array Dequeue is a Dequeue implementation that uses an array for storage.

Code:

```
   import Java. util.*;
   Class Array Dequeue ex {
      Public Static Void main (String args []){
     Array Dequeue< String> dequeue = new Array Dequeue<>();
      dequeue. add ("APPle");
```

```
            deque.add ("Banana");
            system.out.Println (dequeue);
  } }
```

output = [Apple, Banana].

stack = LIFO Implementation of the list Interface.

Code =

```
  import Java.util.*;
   class Stack en {
   Public static void main (String args[]) {
       Stack <string> stack = new stack <> ();
       stack.Push ("Apple");
       stack.Push ("Banana");
       stack.Push ("Cherry");
      System.out.Println (stack);
   }
 }
```

output = [Apple, Banana, Cherry].

vector = A vector is a synchronized implementation of the list Interface.

de =

```
import Java.util.*;
class vector en {
  Public Static Void main (String args [ ]){
    vector < String> vector = new Vector < >( );
      vector.add ("Apple");
      vector.add ("custard Apple");
      System.out.Println (vector);
  }
}
```

otput =

```
[Apple, Custard Apple].
```