Strings

```
public class Main {

    public static void main(String[] args) {

        //String Declaration
        String name = "DIKSHA";
        String fullname = "DIKSHA PATHAK";
        String sentence = "My name is Diksha Pathak";

        //Input from the user
        Scanner sc = new Scanner(System.in);
        //String str=sc.next(); //prints only the first word that is input
        //System.out.println("Your name is : " + str);

        //String str1=sc.nextLine(); //prints the complete sentence that is
input
        //System.out.println("Your name is : " + str1);

        //concatenate the strings
        String firstname = "tony";
        String lastname = "stark";
        String fulln = firstname + "@" + lastname;
        System.out.println("Your name is : " + fulln);

        //length of the string
        System.out.println(fulln.length());

        //charAt - print character by character
        for (int i = 0; i < fulln.length(); i++) {
            System.out.println(fulln.charAt(i));
        }

        //compare two strings
        String name1 = "Tony";
        String name2 = "tony";

        //compareTo returns +ve value if name1>name2 ; negative value if
name1<name2 and 0 if both are equal
        //compareToIgnoreCase ignores the case of the letters
        if (name1.compareToIgnoreCase(name2) == 0) {
            System.out.println("Strings are equal");
        } else {
            System.out.println("Strings are not equal");
        }

        //substring
        String sen = "My name is Tony";
        String str2 = sen.substring(0, 2);
        System.out.println(str2);

        //strings are immutable

        //defining strings from character array
        char[] chars = {'a', 'b', 'c'};
        String st = new String(chars, 1, 2); //output - bc
        System.out.println(st);

        //concatenation with other data types
        String foo = "foobar" + 2 + 2; //foobar22
        System.out.println(foo);
        String foo1 = "foobar" + (2 + 2); //foobar4
        System.out.println(foo1);

        //parsing
        Integer a = 364; // if you need to do type casting, you can only do
via Integer, Long classes and not primitive data types
        String m = a.toString();
        System.out.println(m);
        char[] c = m.toCharArray(); //convert to a character array

        //equals and equalsIgnoreCase- returns true if equal, false otherwise
        String s1 = "hello";
        String s2 = "Hello";
        if (s1.equalsIgnoreCase(s2)) {
            System.out.println("Strings are equal");
        } else {
            System.out.println("Strings are not equal");
        }

        //str.indexOf('t') - returns the first occurence of that character
        //str.lastIndexOf('t') - returns the last occurence of that character

        //str.replace(org,replacement)
        //str.trim(); trims the leading and trailing whitespaces

        //str.toUpperCase() - returns a string
        //str.toLowerCase() - returns a string

    }

    }
```

String Builder

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        //StringBuilder Declaration
        StringBuilder sb = new StringBuilder("Tony");
        System.out.println(sb);

        //char at index 0
        System.out.println(sb.charAt(0));
        //set character at index
        sb.setCharAt(0,'P');
        System.out.println(sb);

        //insert a character
        sb.insert(0,'S');
        System.out.println(sb);

        //delete a part of string
        sb.delete(2,3);
        sb.deleteCharAt(1);
        System.out.println(sb);

        //appending characters
        sb.append(" stark");
        System.out.println(sb);

        //length of the string
        System.out.println(sb.length());

        //reverse a string
        sb.reverse();
        System.out.println(sb);

        //replace
        sb.replace(0,2,"he");
        System.out.println(sb);

    }}
```

Linked List

```
public class Main {

    Node head;
    private int size;

    Main()
    {
        this.size=0;
    }

    class Node {
        String data;
        Node next;

        Node(String data) {
            this.data = data;
            this.next = null;
        }
    }

    //add a node - first
    public void addFirst(String data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            size++;
            return;
        }
        size++;
        newNode.next = head;
        head = newNode;
    }

    //add last
    public void addLast(String data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            size++;
            return;
        }
        size++;
        Node currNode = head;
        while (currNode.next != null) {
            currNode = currNode.next;
        }

        currNode.next = newNode;
    }

    //print
    public void printll() {
        if (head == null) {
            System.out.print("list is empty");
            return;
        }
        Node currNode = head;
        while (currNode != null) {
            System.out.print(currNode.data + " ->");
            currNode = currNode.next;
        }

        System.out.print("Null");
    }

    //delete first
    public void deleteFirst()
    {
        if(head==null)
        {
            System.out.println("list is empty");
            return;
        }
        size--;

        head=head.next;
    }

    //delete last
```

```java
public void deleteLast()
{
    if(head==null)
    {
        System.out.println("list is empty");
        return;
    }
    size--;
    Node prev=head;
    Node curr=head.next;
    if(head.next==null)
    {
        head=null;
        return;
    }

    while(curr.next!=null)
    {
        prev=prev.next;
        curr=curr.next;
    }
    prev.next=null;
}

//return the size
public int getSize()
{
    return size;
}

//reverse the linked list
public void reverseIterate()
{
    if(head==null || head.next==null)
    {
        return;
    }
    Node prev=head;
    Node curr=head.next;
    while(curr!=null)
    {
        Node nextn=curr.next;
        curr.next=prev;

        //update
        prev=curr;
        curr=nextn;
    }
    head.next=null;
    head=prev;
}

public static void main(String[] args) {
    // write your code here


    Main list = new Main();
```

```
        list.addFirst("this");
        list.addFirst("is");
        list.addLast("diksha");
        list.printll();
        System.out.println();
        list.addLast("pathak");
        list.printll();
        System.out.println();
        list.deleteFirst();
        list.printll();
        System.out.println();
        list.deleteLast();
        list.printll();
        int x= list.getSize();
        System.out.println();
        System.out.println(x);

        list.reverseIterate();
        System.out.println();
        list.printll();

    }
}
```

<u>Linked List - Collections Framework</u>

```
import java.util.LinkedList;

public class Main {

    public static void main(String[] args) {
   // write your code here
     LinkedList<String> list = new LinkedList<String>();

     list.addFirst("diksha");
     list.addFirst("pathak");
     System.out.println(list);
     list.addLast("this");
        System.out.println(list);

        System.out.println(list.size());
        for(int i=0; i<list.size(); i++)
        {
            System.out.print(list.get(i)+ " ->");
        }


        System.out.println("NULL");

        list.removeFirst();
        System.out.println(list);

        System.out.println();

        list.removeLast();
        System.out.println(list);

        list.remove(0);
        System.out.println(list);

    }
}
```

Stack implementation using LinkedList

```
public class Main {

    static class Node{
        int data;
        Node next;
        public Node(int data)
        {
            this.data=data;
            next=null;
        }
    }

    static class Stack{
        public static Node head;

        public static boolean isEmpty()
        {
            if(head==null)
            {
                return true;
            }
            return false;
        }
        public static void push(int data)
        {
            Node newnode=new Node(data);

            if(isEmpty())
            {
                head=newnode;

            }
            newnode.next=head;
            head=newnode;
        }

        public static int pop()
        {
            if(isEmpty())
            {
                return -1;
            }

            int top=head.data;
            head=head.next;
            return top;
        }

        public static int peek()
        {
            if(isEmpty())
            {
                return -1;
            }


            return head.data;
        }
    }

    public static void main(String[] args) {
        // write your code here
        Stack s =new Stack();

        s.push(1);
        s.push(2);
        s.push(3);
        s.push(4);
        s.push(5);

        while(!s.isEmpty()){
            System.out.println(s.peek());
            s.pop();
        }



    }
}
```

<u>Stack using Array Lists</u>

```
import java.util.ArrayList;

public class Main {

   static class Stack{
       static ArrayList<Integer> list = new ArrayList<>();

        public static boolean isEmpty()
        {
            return list.size()==0;
        }

        public static void push(int data)
        {
            list.add(data);
        }

        public static int pop()
        {
            if(isEmpty())
            {
                return -1;

            }
            int top=list.get(list.size()-1);
            list.remove(list.size()-1);

            return top;
        }

        public static int peek()
        {
            if(isEmpty())
                return -1;
            return list.get(list.size()-1);
        }
    }

    public static void main(String[] args) {
        // write your code here
        Stack s =new Stack();

        s.push(1);
        s.push(2);
        s.push(3);
        s.push(4);
        s.push(5);

        while(!s.isEmpty()){
            System.out.println(s.peek());
            s.pop();
        }}}
```

Stack using Collections framework

```
import java.util.ArrayList;
import java.util.Stack;

public class Main {

    public static void main(String[] args) {
        // write your code here
        Stack<Integer> s=new Stack<>();


        s.push(1);
        s.push(2);
        s.push(3);
        s.push(4);
        s.push(5);

        while(!s.isEmpty()){
            System.out.println(s.peek());
            s.pop();
        }}}
```

Sorting

Bubble Sort

```
    public static void main(String[] args) {

        int[] arr={7,8,3,1,2};
        int temp;

        //time complexity - O(n^2)
        //bubble sort
        for(int i=0; i<arr.length-1; i++)
        {
            for(int j=0; j<arr.length-i-1; j++)
            {
                if(arr[j]>arr[j+1])
                {
                    temp=arr[j];
                    arr[j]=arr[j+1];
                    arr[j+1]=temp;
                }
            }

        }}
```

Selection Sort

```
    public static void main(String[] args) {

        int[] arr={7,8,3,1,2};
        int min,temp;

        //time complexity - O(n^2)
        //selection sort
        for(int i=0; i<arr.length-1; i++)
        {
            min=i;
            for(int j=i+1; j<arr.length; j++)
            {
                if(arr[j]<arr[min])
                {
                    min=j;
                }
            }
            temp=arr[i];
            arr[i]=arr[min];
            arr[min]=temp;
        }}
```

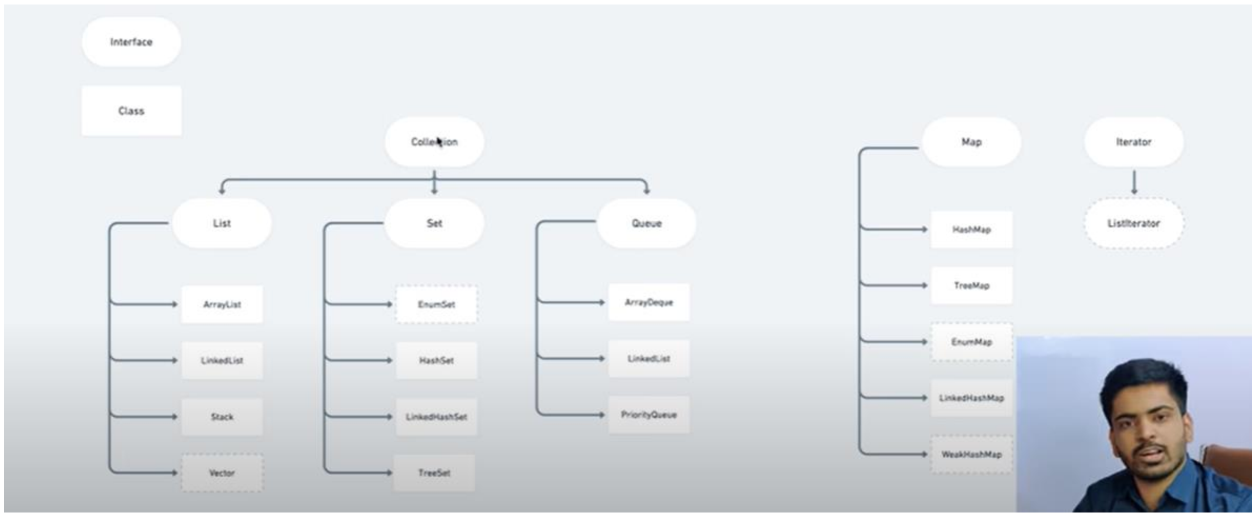
Insertion Sort

```
public static void main(String[] args) {
        int[] arr={7,8,3,1,2};
        //time complexity - O(n^2)
        //insertion sort
        for (int i = 1; i < arr.length; ++i) {
            int key = arr[i];
            int j = i - 1;

            /* Move elements of arr[0..i-1], that are
               greater than key, to one position ahead
               of their current position */
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
}
```

Arrays Class

```
import java.util.Arrays;

public class LearnArrayClass {

    public static void main(String[] args) {
        Integer[] nums={1,2,3,4,5,6,7,8,9,10};

        int index= Arrays.binarySearch(nums,10);
        System.out.println("Element found at : " + index);

        //if array is not sorted
        Arrays.sort(nums); //implement quick sort

        //there is a parallel sort also when your array has many numbers

        Arrays.fill(nums,9);
       for(int i:nums) {
           System.out.println(nums[i]);
       }

    }
}
```

Collections Framework



ArrayLists

```java
import java.util.ArrayList;
import java.util.Collections;

public class Main {

    public static void main(String[] args) {
        ArrayList<Integer> list=new ArrayList<>();
        ArrayList<String> list2= new ArrayList<String>();
        ArrayList<Boolean> list3 = new ArrayList<>();


        //add elements
        list.add(0);
        list.add(2);
        list.add(3);
        System.out.println(list);

        //get elements
        int element = list.get(0);
        System.out.println(element);

        //add element in between
        list.add(1,1);
        System.out.println(list);

        //set elements
        list.set(0,5);
        System.out.println(list);

        //delete element
        list.remove(3);
        System.out.println(list);

        //size
        int s = list.size();
        System.out.println(s);
        //loops
        for(int i=0; i<s; i++)
        {
            System.out.println(list.get(i));
        }
        System.out.println();

        //sorting
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class LearnQueue {

    public static void main(String[] args) {
        Queue<Integer> qu = new LinkedList<>();

        //add element
        qu.offer(12);
        qu.offer(13);
        qu.offer(14);
        qu.offer(15);
        System.out.println(qu);
        //remove element
        System.out.println(qu.poll());

        //top element
        System.out.println(qu.peek());

    }

}
```

ArrayDequeue

```
import java.util.ArrayDeque;

public class LearnArrayDequeue {

    public static void main(String[] args) {
        ArrayDeque<Integer> adq=new ArrayDeque<>();

        adq.offer(123);
        adq.offerFirst(20);
        adq.offerLast((10));
        System.out.println(adq);
        System.out.println(adq.stream().findFirst());
        for (Integer item: adq) {
        }
        System.out.println(adq.peek());
        System.out.println(adq.peekFirst());
        System.out.println(adq.peekLast());

        System.out.println(adq.poll());
        System.out.println(adq.pollFirst());
        System.out.println(adq.pollLast());

        System.out.println(adq);
    }

}
```

Priority Queue

```
import java.util.Comparator;
import java.util.PriorityQueue;

public class LearnPriorityQueue {
    public static void main(String[] args) {
        //reversing the order
        //increasing order - min heap
        //decreasing order - max heap
        PriorityQueue<Integer> pq= new
PriorityQueue<>(Comparator.reverseOrder());

        pq.offer(40);
        pq.offer(12);
        pq.offer(10);
        pq.offer(36);

        System.out.println(pq);

        pq.poll();
        System.out.println(pq);

        System.out.println(pq.peek());

    }

}
```

HashSet

```
public class LearnHashSet {

    public static void main(String[] args) {

        //Set<Integer> set = new HashSet<>(); //all elements are unique in
hashset - O(1)
        //Set<Integer> set = new LinkedHashSet<>(); //linked hash set
        //all the properties are same except now elements are added in an
arranged manner
```

```
        Set<Integer> set = new TreeSet<>(); //-o(log n)
        set.add(20);
        set.add(3);
        set.add(50);
        set.add(44);
        set.add(19);
        set.add(50); //already added so it wont get added again

        System.out.println(set);

        set.remove(44);
        System.out.println(set);

        System.out.println(set.contains(100));

        System.out.println(set.isEmpty());

        System.out.println(set.size());

        set.clear();

        System.out.println(set);

    }
}
```

<u>Map</u>

```
import com.sun.source.tree.Tree;

import java.util.*;

public class LearnMap {

    public static void main(String[] args) {
        //Map<String, Integer> numbers = new HashMap<>(); - O(1)
        Map<String, Integer> numbers = new TreeMap<>(); //-O(log n)
        numbers.put("one",1);
        numbers.put("two",2);
        numbers.put("three",3);
        numbers.put("four",4);


        System.out.println(numbers);

        if(!numbers.containsKey("three"))
        {
            numbers.put("three",33);
        }

        System.out.println(numbers.containsValue(3));
        System.out.println(numbers.isEmpty());

        numbers.putIfAbsent("five",5);

        System.out.println(numbers);

        //iterate through the map
        for(Map.Entry<String,Integer> e: numbers.entrySet())
        {
            System.out.println(e);
            System.out.println(e.getKey());
            System.out.println(e.getValue());

        }

        for (String key: numbers.keySet())
        {
            System.out.println(key);
        }

        for(Integer value: numbers.values())
        {
            System.out.println(value);
        }



    }
}
```

<u>Collections Class</u>

```
public class LearnCollectionsClass {

    public static void main(String[] args) {
        List<Integer> list=new ArrayList<>();

        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        list.add(60);
        list.add(20);
        list.add(20);

        System.out.println("Minimum element of the arraylist is : " +
Collections.min(list));
        System.out.println("Maximum element of the arraylist is : " +
Collections.max(list));
        System.out.println(Collections.frequency(list,20));

       // Collections.sort(list, Comparator.reverseOrder());
        Collections.sort(list);
        System.out.println(list);
    }
}
```

<u>Queues using Array</u>

```
public class QueueScratch {
 static class queues{
```

```java
static int[] arr;
static int size;
static int rear=-1;

queues(int size)
{
    arr=new int[size];
    this.size=size;
}

public static boolean isEmpty()
{
    return rear==-1;
}

//add function
public static void add(int data)
{
    if(rear==size-1)
    {
        System.out.println("Queue is full");
        return;
    }
    rear++;
    arr[rear]=data;
}

//delete function - O(n)
public static int remove()
{
    if(isEmpty())
    {
        System.out.println("Queue is empty");
        return -1;
    }

    int front=arr[0];

    for(int i=0; i<rear; i++)
    {
        arr[i]=arr[i+1];
    }
    rear=rear-1;
    return front;
}

//peek
public static int peek()
{
    if(isEmpty())
    {
        System.out.println("Queue is empty");
        return -1;
    }

    return arr[0];
}
```

```java
 }

    public static void main(String[] args) {

     queues q=new queues(10);
     q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);

        while(!q.isEmpty())
        {
            System.out.println(q.peek());
            q.remove();
        }


    }

}
```

Circular Queues using Array

```java
public class CircularQueue {

    static class cqueue{
        static int[] arr;
        static int size;
        static int front=-1;
        static int rear=-1;

        cqueue(int size)
        {
            arr=new int[size];
            this.size=size;
        }

        public static boolean isEmpty()
        {
            return rear==-1 && front==-1;
        }

        public static boolean isFull()
        {
            return (rear+1)%size==front;
        }

        //add
        public static void add(int data)
        {
            if(isFull())
            {
                System.out.println("Queue is full");
                return;
            }
            //if empty
            if(front==-1)
            {
                front=0;
            }
            rear=(rear+1)%size;
            arr[rear]=data;

        }

        //delete an element
        public static int remove()
        {
            if(isEmpty())
            {
                System.out.println("Queue is empty");
                return -1;
            }

            int result=arr[front];

            if(rear==front)
            {
                rear=front=-1;
            }
            else
            {
                front=(front+1)%size;
            }

            return result;
        }

        //peek
        public static int peek()
        {
            if(isEmpty())
            {
                System.out.println("Queue is empty");
                return -1;
            }

            return arr[front];
        }



    }
    public static void main(String[] args) {

        cqueue q=new cqueue(5);
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        q.add(5);
        System.out.println(q.remove());
```

```
        q.add(6);
        System.out.println(q.remove());
        q.add(7);

        while(!q.isEmpty())
        {
            System.out.println(q.peek());
            q.remove();
        }}}
```

Queue using LinkedList

```
public class QueueLL {

    static class Node
    {
      static  int data;
      static  Node next;

        Node(int data)
        {
            this.data=data;
            next=null;
        }
    }
    static class llqueue{
      static Node head=null;
      static Node tail=null;

        public static boolean isEmpty()
        {
            return head==null && tail==null;
        }

        //add
        public static void add(int data)
        {
           Node newnode=new Node(data);
            //if empty
            if(tail==null)
            {
               tail=head=newnode;
               return;
            }
            tail.next=newnode;
            tail=newnode;

        }

        //delete an element
        public static int remove()
        {
            if(isEmpty())
            {
                System.out.println("Queue is empty");
                return -1;
            }

            int front=head.data;
            if(head==tail)
            {
                tail=null;
            }
            head=head.next;
            return front;
        }

        //peek
        public static int peek()
        {
            if(isEmpty())
            {
                System.out.println("Queue is empty");
                return -1;
            }

            return head.data;
        }}
    public static void main(String[] args) {

        llqueue q=new llqueue();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        q.add(5);
        System.out.println(q.remove());
        q.add(6);
        System.out.println(q.remove());
        q.add(7);

        while(!q.isEmpty())
        {
            System.out.println(q.peek());
            q.remove();
        }

    }}
```