*1. Check number is* ***Odd & Even****?*

*2. Check number is* ***Prime*** *or Not(decided by 1 or itself like 3, 7, 13, 17, 23, 11)?*

*8. check* ***number*** *is* ***Armstrong*** *or not(153=> 1x1x1 + 5x5x5 + 3x3x3 = 153, 370=> 3x3x3 + 7x7x7 + 0x0x0 = 370)?*

*10. Check Number is* ***Positive or Negative****? (0,1, 2>.. Positive and -1,-2< is Negative)*

*3.* ***Reverse*** *Number using* ***while loop***

*4.* ***Reverse*** *Number using* ***recursion***

*5.* ***Factorial*** *of a Number(get all possible number by which given number will divid)*

*6.* ***Reverse*** *a string using* ***reverse****() method with strinBuilder class*

*7.* ***Reverse*** *a string using* ***for loop*** *with convert string into toCharArray*

*9. Print* ***alphabet*** *using for loop*

*11.* ***Swap*** *two numbers using* ***temporary/third*** *variable*

*12.* ***Swap*** *two numbers* ***without*** *using* ***temporary/third*** *variable*

*13. Check a* ***Leap*** *Year?*

*14. Check* ***string*** *is* ***Palindrome*** *or Not?*

*14. Check* ***Number*** *is* ***Palindrome*** *or Not?*

*15. Display* ***Fibonacci*** *Series Using* ***for Loop***

***16.*** *How to check Given input is String or Integer.*

**Searching**

1. **Sequential/Liner Search** =>( Applied on unsorted Array) => using for loop
2. **Binary/Interval Search**=>(Applied on sorted array) => using while loop with start, end and mid index

**Sorting**

**1. Selection Sort:**

**1. Bubble Sort:**

**2. Marge Sort:**

**3. Quick Sort:**

**Duplicate item**

1. **Remove duplicate items**
2. **Find duplicate items**
3. **Find & Count the duplicate items**

**Difference**

1. **List and Set**
2. **Array and ArrayList**
3. **ArrayList and LikedList**
4. **LikedList and LinkedHashSet**
5. **HashSet and LinkedHashSet**
6. **HashSet and HashMap**

**7. HashMap and HashTable**

***//given number is even or odd***

void isOddEvenNumber(){

*// 2, 4, 50, 100 : Even Number*

*// 3, 5, 7, 11: Odd Number*

int actualNumber = 153;

int oddEven = actualNumber % 2;

if(oddEven == 0){

System.*out*.println("Even Number");

}

else{

System.*out*.println(“Odd Number");

}

}

***//Check Whether a Number is Prime or Not***

*// which number can’t be divide any number like 7, 13, 17*

boolean isNumberPrime(){

int number = 13;

boolean isPrime = true;

if(number ==0 || number == 1){

isPrime = false;

}else{

for(int i=2; i<number; i++){

if(number%i == 0){

isPrime = false;

break;

}

}

if(isPrime){

System.*out*.println("Given "+number+" is prime");

}else{

System.*out*.println("Given "+number+" is not prime");

}

}

return isPrime;

}

***// Reverse a number using while loop***

void getReverseNumber(){

int number = 987654, reverse = 0;

if(number <10){

System.*out*.println("The reverse of the given number is: "+ number);

}else{

while(number != 0) {

int remainder = number % 10; *// 4, 5, 6, 7, 8, 9*

reverse = reverse \* 10 + remainder; *//4, 45, 456,4567,45678, 456789*

number = number/10; *// 98765, 9876, 987, 98, 9, 0*

}

System.*out*.println("The reverse of the given number is: "+ reverse);

}

}

***// Reverse a number using recursion***

static int *reverseNumber* = 0;

public static void reverseMethod(int number) {

if (number <10) {

int module = number % 10;

*reverseNumber* = *reverseNumber* \* 10 + module;

System.*out*.println("reverseNumber: "+ *reverseNumber*);

return;

}

else {

int module = number % 10;

*reverseNumber* = *reverseNumber* \* 10 + module;

int newNumber = number/10;

System.*out*.println("reverNumberX: "+ *reverseNumber*);

System.*out*.println("newNumber: "+ newNumber);

*//Method is calling itself: recursion*

*reverseMethod*(newNumber);

}

}

***//factorial of a Number***

*// get all possible number by which given number will divid*

void getFectorialsOfNumber(){

int number = 50;

System.*out*.print("Factors of "+ number + " are: ");

*// loop runs from 1 to number*

for (int i = 1; i <= number; ++i) {

*// if number is divided by i(1,2, 3, 4..) then i is the factor that number*

if (number % i == 0) {

System.*out*.print(i + "");

}

}

}

***Reverse String=> Reverse(), StringBuilder(), Concatinate, charArry***

***// Reverse a string using reverse() method of the StringBuilder class***

String getReverseString() {

     String str = "Abhishek"; // kehsihbA

     char arr[] = new char[str.length()];

     for(int i = arr.length-1; i>=0; i--){

         arr[arr.length - i -1] = str.charAt(i);

    }

    String revVal = String.valueOf(arr);

     System.out.println("reverse String is "+revVal);

}

void reverseString(){

String inputString = "Geeks for Geeks";

StringBuilder inputSB = new StringBuilder();

inputSB.append(inputString);

inputSB.reverse();

System.*out*.println(inputSB);

}

***// Reverse a string using toCharArray() method***

void reverseString(){

String input = "GeeksForGeeks";

StringBuilder Sb = new StringBuilder();

char[] charArr = input.toCharArray();

for (int i = charArr.length - 1; i >= 0; i--) {

Sb.append(charArr[i]);

System.*out*.print(charArr[i]);

}

}

***//Swap two numbers using temporary/third variable***

void swapTwoNumber(){

int firstNumber = 10, secondNumber = 20;

System.*out*.println("Before swap firstNumber:"+ firstNumber +" secondNumber:"+secondNumber);

int tempVal = firstNumber;

firstNumber = secondNumber;

secondNumber = firstNumber;

System.*out*.println("After swap firstNumber:"+ firstNumber +" secondNumber:"+secondNumber);

}

***//Swap two numbers without using third variable***

void swapTwoNumber(){

int firstNumber = 10, secondNumber = 20;

firstNumber = secondNumber + firstNumber; *// 10+20 = 30*

secondNumber = firstNumber - secondNumber; *// 30-20 = 10*

firstNumber = firstNumber - secondNumber; *// 30 -10 = 20*

}

***//Check a Leap Year***

void isLeapYearOrNot(){

int year = 2012;

boolean isLeapYear = false;

if(year % 4 == 0){

if(year % 100 ==0){

if(year % 400 == 0){

isLeapYear = true;

}else{

isLeapYear = false;

}

}else{

isLeapYear = true;

}

}else{

isLeapYear = false;

}

if(isLeapYear){

System.*out*.println("Yes leap Year");

}else{

System.*out*.println("Not a leap Year");

}

}

***//given number is armstrong number or not***

void isArmstrongNumber(){

*//153: 1^3 + 5^3 + 3^3 = 1 + 125+ 27 = 153 (an Armstrong Number)*

*//125: 1^3 + 2^3 + 5^3 = 1 + 8 + 125 = 134 (Not an Armstrong Number)*

int actualNumber = 155;

int tempNumber = actualNumber;

int subOfNum = 0;

int totalDigit = String.*valueOf*(actualNumber).length();

while(tempNumber!=0) {

int module = tempNumber % 10;

int modulePowModule = (int) Math.*pow*(module, totalDigit); *//temp\*temp\*temp;*

subOfNum = subOfNum + modulePowModule;

tempNumber = tempNumber / 10;

}

if(subOfNum == actualNumber) {

System.*out*.println(actualNumber + " is an Armstrong number");

}

else {

System.*out*.println(actualNumber + " is not an Armstrong number");

}

}

***//Display uppercased alphabet using for loop***

*//A B C D E F G H I J K L M N O P Q R S T U V W X Y Z*

void printAlphabetAZ(){

for(char c = 'A'; c <= 'Z'; ++c){

System.*out*.print(c + "");

}

}

***//Check if a Number is Positive or Negative using if else***

void isPositiveOrNegiveNumber(){

double number = 12.3;

if (number <0.0) {

*// true if number is less than 0*

System.*out*.println(number + " is a negative number.");

} else if ( number >0.0) {

*// true if number is greater than 0*

System.*out*.println(number + " is a positive number.");

} else {

*// if both test expression is evaluated to false*

System.*out*.println(number + " is 0.");

}

}

***//Palindrome means revers of string/number are equal to actual number***

*// Palindrome==> radar <=> radar*

*// non Palindrome ==> java <=> java*

void isPalindromeString(){

String str = "java";

StringBuffer newString = new StringBuffer(str);

newString.reverse();

String reverseString = newString.toString();

if(str.equalsIgnoreCase(reverseString)){

System.*out*.println("Given String is Palindrome");

}else{

System.*out*.println("Given String is Not Palindrome");

}

}

*//****Palindrome*** *means revers of string/number are equal to actual number*

*// Palindrome==> 121 <=> 121*

*// non Palindrome ==> 122 <=> 221*

void isPalindromeNumber(){

int num = 121;

int actualNum = num;

int newNum = 0;

while (num != 0){

int mod = num % 10;

newNum = newNum \* 10 + mod;

num = num / 10;

}

if(actualNum == newNum){

System.*out*.println("Given Number is Palindrome");

}else{

System.*out*.println("Given Number is Not Palindrome");

}

}

***//Fibonacci Series Using for Loop***

*//Fibonacci Series: 5, 10, 15, 25, 40, 65, 105, 170, 275, 445,*

void getFibonacciSeries(){

int number = 10;

int firstTerm = 5, secondTerm = 10;

System.*out*.println("Fibonacci Series till "+ number + " terms:");

for (int i = 1; i <= number; ++i) {

System.*out*.print(firstTerm + ", ");

*// compute the next term*

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

**// *check Given input is String or Integer***

**voidcheckValueIsNumberString(){**

    String str = "6790";

**if**(isNumber(str)){

            System.out.println(“Integer");

}

**else{**

            System.out.println(“String");

}

   }

// Returns true if s is a number else false

**booleanisNumber(String s){**

**for**(**int**i = 0; i < s.length(); i++){

**if**(Character.isDigit(s.charAt(i)) == **false**){

**returnfalse**;

  }

**returntrue**;

  }

booleanisNumber(String s){

try{

Long.parseLong(s);

}

catch(NumberFormatException ex){

return false;

}

return true;

}

**Search ing:**

* Searching is a process of finding a particular element among several given elements.
* The search is successful if the required element is found.
* Otherwise, the search is unsuccessful.

**Sequential/Liner Search => (https://www.gatevidyalay.com/linear-search-searching-algorithms/)**

**(Apply on unsorted Array) => using for loop**

* Linear Search is the simplest searching algorithm.
* It traverses the array sequentially to locate the required element.
* It searches for an element by comparing it with each element of the array one by one.
* So, it is also called as **Sequential Search**.

**Time Complexity** = Best Case **O(1)** and Worst Case **O(n)**

*// Search Item with integer aray*

void LinerSearch(){

int arr[] = {3, 1, 11, 10, 13, 4, 50, 5};

int findItem = 10;

for(int i=0; i <arr.length; i++){

if(arr[i] == findItem){

System.*out*.println("Item Index is:"+i +" and item is "+findItem);

}

}

}

*//****Liner Search*** *Item with String aray*

void LinerSearch(){

String arr[] = {"Ram", "Shayam", "Geeta", "Sheeta"};

String findItem = "Geeta";

for(int i=0; i <arr.length; i++){

if(arr[i].equals(findItem)){

System.*out*.println("Item Index is:"+i +" and item is "+findItem);

}

}

}

*//****Liner Search*** *Item with String aray*

*//The* ***compareTo()*** *method compares two strings* ***lexicographically****. ... The method returns 0 if the string is equal to the other string. A value less than 0 is returned if the string is less than the other string (less characters) and a value greater than 0 if the string is greater than the other string (more characters).*

void LinerSearch(){

String arr[] = {“Abhi”,"Ram","Shayam","Geeta", “Shatish”};

String findItem = "Geeta";

for(int i=0; i <arr.length; i++){

System.*out*.println("Item Index is:"+ i + " and item is "+ arr[i].compareTo(findItem));

if(arr[i].compareTo(findItem) == 0) {

System.*out*.println("Item Index is:"+ i + " and item is "+ findItem);

}

}

}

**Search ing: Binary Search: it’s (https://www.gatevidyalay.com/binary-search-binary-search-algorithm/)**

Binary search is a fast search algorithm with run-time complexity of Ο(log2 n) and best TC is O(1) for sorted item collection. This search algorithm works on the principle of divide and conquer.

* Binary Search is one of the fastest searching algorithms.
* It is used for finding the location of an element in a linear array.
* It works on the principle of divide and conquer technique

*// Binary Search Item with int aray*

void BinarySearch(){

int[] arr = {2,7, 11, 16,49, 55};

int findItem = 49;

int firstIndex = 0;

int lastIndex= arr.length -1;

int midIndex = (firstIndex + lastIndex) / 2;

while (firstIndex < lastIndex){

if(findItem == arr[midIndex]){

System.*out*.println("Find Item:"+findItem +" and index:"+ midIndex);

break;

}else if(findItem <arr[midIndex] ){

lastIndex = midIndex -1;

}else{

firstIndex = midIndex +1;

}

midIndex = (firstIndex + lastIndex) / 2;

}

}

*// Binary Search Item with int aray*

void BinarySearch(){

String[] arr = {"Abhi","Geeta", "Hitesh", "Indra", "Ram", "Shayam","Sheeta"};

String findItem = "Ram";

int firstIndex = 0;

int lastIndex= arr.length -1;

int midIndex = (firstIndex + lastIndex) / 2;

while (firstIndex <= lastIndex){

int comPareVal = findItem.compareTo(arr[midIndex]);

System.*out*.println("Item:"+lastIndex +" comPareVal:"+comPareVal);

if(comPareVal == 0){

System.*out*.println("Find Item:"+findItem +" and index:"+ midIndex);

break;

}else if(comPareVal <0){

lastIndex = midIndex -1;

System.*out*.println("Item not found and less "+lastIndex);

}else{

firstIndex = midIndex +1;

System.*out*.println("Item not found and gretter "+lastIndex);

}

midIndex = (firstIndex + lastIndex) / 2;

}

}

**Sorting algorithm**

*//In* ***selection sort algorithm****, we search for the lowest element and arrange it to the proper location. We swap the current element with the next lowest number and* time complexity is O(n2).

void SalectionSort(){

int arr[] = {2, 50, 1, 20, 4, 3};

System.*out*.println("Before Sorting: "+ Arrays.*toString*(arr));

for(int i=0; i<arr.length; i++){

int minItemIndex = i;

for(int j = i+1; j<arr.length; j++){

if(arr[j] <arr[minItemIndex]){

minItemIndex = j;

}

}

int temp = arr[minItemIndex];

arr[minItemIndex] = arr[i];

arr[i] = temp;

System.*out*.println("In Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("After Sorting: "+ Arrays.*toString*(arr));

*/\**

*Before Sorting: [2, 50, 1, 20, 4, 3]*

*In Sorting: [1, 50, 2, 20, 4, 3]*

*In Sorting: [1, 2, 50, 20, 4, 3]*

*In Sorting: [1, 2, 3, 20, 4, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*After Sorting: [1, 2, 3, 4, 20, 50]*

*\*/*

}

*//***Bubble Sort:** *Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order.* time complexity is O(n).

void BubbleSort(){

int arr[] = {2, 1, 4, 3, 0, -1};

for(int i=0; i<arr.length; i++){

for(int j = i+1; j<arr.length; j++){

if(arr[j] <arr[i]){

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

System.*out*.println("Inner In Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("Inner Out Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("After Sorting: "+ Arrays.*toString*(arr));

*/\**

*Before Sorting: [2, 1, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [0, 2, 4, 3, 1, -1]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner Out Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 1, 4, 3, 2, 0]*

*Inner In Sorting: [-1, 0, 4, 3, 2, 1]*

*Inner Out Sorting: [-1, 0, 4, 3, 2, 1]*

*Inner In Sorting: [-1, 0, 3, 4, 2, 1]*

*Inner In Sorting: [-1, 0, 2, 4, 3, 1]*

*Inner In Sorting: [-1, 0, 1, 4, 3, 2]*

*Inner Out Sorting: [-1, 0, 1, 4, 3, 2]*

*Inner In Sorting: [-1, 0, 1, 3, 4, 2]*

*Inner In Sorting: [-1, 0, 1, 2, 4, 3]*

*Inner Out Sorting: [-1, 0, 1, 2, 4, 3]*

*Inner In Sorting: [-1, 0, 1, 2, 3, 4]*

*Inner Out Sorting: [-1, 0, 1, 2, 3, 4]*

*Inner Out Sorting: [-1, 0, 1, 2, 3, 4]*

*After Sorting: [-1, 0, 1, 2, 3, 4]*

*\*/*

}

**//Marge Sort: The merge sort algorithm is based on the principle of divide and conquer algorithm where a problem is divided into multiple sub-problems. Each sub-problem is solved individually and finally, sub-problems are combined to form the final solutions.time complexity is O(n).**

void margeSort(){

int[] array = {11, 7, 3, 2, 6, 5, 12, 10, 9, 1,0 };

divideIntoSubArr(array, 0, array.length - 1);

}

void divideIntoSubArr(int[] arr, int leftIndex, int rightIndex) {

if(leftIndex < rightIndex){

*// midIndex is the point where the array is divided into two sub arrays*

int midIndex = (leftIndex + rightIndex) / 2;

*// recursive call to each sub arrays*

divideIntoSubArr(arr, leftIndex, midIndex);

divideIntoSubArr(arr, midIndex + 1, rightIndex);

*// Merge the sorted sub arrays*

mareSortedArr(arr, leftIndex, midIndex, rightIndex);

}

}

void mareSortedArr(int[] array, int leftIndex, int midIndex, int ightIndex) {

int LAS = midIndex - leftIndex + 1;

int RAS = rightIndex - midIndex;

int LA[] = new int[LAS];

int RA[] = new int[RAS];

for (int i = 0; i <LAS; i++){

LA[i] = array[leftIndex + i];

}

for (int j=0; j<RAS; j++){

RA[j] = array[midIndex + 1 + j];

}

*// Maintain current index of sub-arrays and main array*

*// Until we reach either end of either L or M, pick larger among*

*// elements L and M and place them in the correct position at A[p..r]*

*// for sorting in descending*

*// use if(L[i] >= <[j])*

int i, j, k;

i = 0;

j = 0;

k = leftIndex;

while (i <LAS && j <RAS){

if( LA[i] <= RA[j] ){

array[k] = LA[i];

i++;

}else{

array[k] = RA[j];

j++;

}

k++;

}

*// When we run out of elements in either L or M,*

*// pick up the remaining elements and put in A[p..r]*

while ( i <LAS){

array[k] = LA[i];

i++;

k++;

}

while ( j <RAS){

array[k] = RA[j];

j++;

k++;

}

System.*out*.println("Sorted Arrr:"+Arrays.*toString*(array));

}

*/\*Before Sorting:= [11, 7, 3, 2, 6, 5, 12, 10, 9, 1,0 ];*

*Sorted Arrr:[7, 11, 3, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 5, 6, 12, 10, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 12, 10, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 10, 12, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 9, 10, 12, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 9, 10, 12, 0, 1]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 0, 1, 9, 10, 12]*

*Sorted Arrr:[0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 12]*

*\*/*

Find **Duplicate items in Array & ArrayList**

List<String>aList = new ArrayList<>();

aList.add("Abhishek");

aList.add("Chandan");

aList.add("Rai");

aList.add("Abhishek");

aList.add("Rudra");

aList.add("Abhishek");

aList.add("Rudra");

aList.add("Abhishek");

String[] arr = {"Chandan", "Rudra", "Rai", "Rudra", "Rudra", "Rai"};

*// List<String> list = new ArrayList<String >(Arrays.asList(arr));*

*// Set<String> uniqList = new HashSet<>(aList);*

*// Set<String> uniqLis = new HashSet<>(Arrays.asList(arr));*

aList.addAll(Arrays.*asList*(arr));

*//****Find duplicate*** *items Using second ArrayList & check items is contain or not*

List<String>uniqArrayList = new ArrayList<>();

for (String item: aList) {

if(uniqArrayList.contains(item)){

System.*out*.println("duplicate Items found in AL:"+item);

}else{

uniqArrayList.add(item);

}

}

*/\* duplicate Items found in AL:Abhishek*

*duplicate Items found in AL:Abhishek*

*duplicate Items found in AL:Rudra*

*duplicate Items found in AL:Abhishek*

*\*/*

*//****Remove duplicate items*** *|| get only unque item Using HashSet and add items at time of inialization or addAll methos*

Set<String>uniqSet = new HashSet<>(aList);

System.*out*.println("Unique items List :"+uniqSet);

*//uniqSet.addAll(aList);*

*/\**

*Unique items List :[Abhishek, Rudra, Rai, Chandan]*

*\*/*

*//****Remove duplicate items*** *Using others LinkedHashSet & check items add or not in list*

Set<String>hashSet = new HashSet<>();

for (String item: aList) {

boolean isAdd = hashSet.add(item);

if(!isAdd){

System.*out*.println("Duplicate Items found using LHS :"+item);

}

}

*/\* Duplicate Items found using LHS :Abhishek*

*Duplicate Items found using LHS :Abhishek*

*Duplicate Items found using LHS :Rudra*

*Duplicate Items found using LHS :Abhishek*

*\*/*

*//****find & Count the duplicate*** *items Using HashMap & check items is contain(if contain count++) or if not(count 1)*

HashMap<String,Integer>hashMap = new HashMap<>();

for(String i: aList) {

if (hashMap.containsKey(i)) {

hashMap.put(i, hashMap.get(i) + 1);

}else {

hashMap.put(i, 1);

}

}

System.*out*.println("HashMap with Duplicate count:"+ hashMap);

*/\**

*HashMap with Duplicate count:{Abhishek=4, Rudra=2, Rai=1, Chandan=1}*

*\*/*

*//find & Count the duplicate items Using HashMap & check items is contain(if contain count++) or if not(count 1)*

*// Print it in custome tag*

HashMap<String,Integer>hashMap1 = new HashMap<>();

for(String i: aList) {

if (hashMap1.containsKey(i)) {

hashMap1.put(i, hashMap1.get(i) + 1);

}else {

hashMap1.put(i, 1);

}

}

//for(Map.Entry<String, Integer>mapx :hashMap.entrySet()){

for(HashMap.Entry<String, Integer>mapx :hashMap.entrySet()){

String key = mapx.getKey();

Integer count = mapx.getValue();

System.*out*.println("HashMap with Duplicate Items:"+key + " count:"+count);

}

*/\* HashMap with Duplicate Items:Abhishek count:4*

*HashMap with Duplicate Items:Rudra count:2*

*HashMap with Duplicate Items:Rai count:1*

*HashMap with Duplicate Items:Chandan count:1*

*\*/*

**Diff b/w Collection and Collections:**

**Collection:**it is an Interface, where no of interface are integrate(list,set, Que). It is collection of individual object which represent a single entity is know as collection.

**list**: ArrayList, LinkedList, Vector, Stack

S**et**:HashSet, LinkedHashSet

**Que:** PriorityQue, Deque, LinkedList

**Collections:** this is an utility class which defined several utility method for collection object like (Collections.sor(list), Collections.sort.reverseorder(ar))

**Difference between List and Set:**

| **Property** | **List** | **Set** |
| --- | --- | --- |
| Use | if you want to store group of object into single entity where duplicate element are allowed and most be follow insertion order | if you want to store group of object into single entity where duplicate element are not allowed and most be follow insertion order are not required |
| **Order** | List is an insertion /ordered sequence. | Set is an unordered sequence or doesn’t maintained any order |
| **Duplicate** | List allows any number of duplicate elements | Set doesn’t allow duplicate elements |
| **Null** | List allows any number of null elements | Set allow only one null elements |
| **Class/Implemantion** | List implementations are ArrayList, LinkedList, Vector, Stack | Set implementations are HashSet, LinkedHashSet. |
| **Use** | It is used when we want to frequently access the elements by using the index. | it is used when we frequently need to perform crude operation |
| **Access** | Elements by their position can be accessed.  We can get the element of a specified index from the list using the get() method. | Position access to elements is not allowed.  We cannot find the element from the Set based on the index because it doesn't provide any get method(). |
|  |  |  |

**Difference between Array and ArrayList:**

| **Property** | **Array** | **ArrayList** |
| --- | --- | --- |
| **Dimensionality** | It can be single-dimensional or multidimensional | It can only be single-dimensional |
| **Declaration** | String[] arr = {}; | ArrayList<String> obj = new ArrayList<String>(); |
| **Size** | Array is static in size that is fixed length data structure, One declare we can not change the length after creating the Array object | ArrayList is dynamic in size. As elements are added to an ArrayList its capacity grows automatically |
| **Traversing Elements** | **For** and **foreach** are generally used for iterating over arrays | Here **iterator** and **foreach** are used to traverse the ArrayList |
| **Length** | length keyword can give the total size of the array | size() method is used to compute the size of ArrayList |
| **Speed** | It is faster as above we see it of fixed size | It is relatively slower because of its dynamic nature |
|  | Array can contain both primitive/non primitive data types((like int , float , double) ) as well as objects. | ArrayList can not contains primitive data types & it can contains only Object |
|  | performance wise best but memories wise not best | memories wise best but performance wise not best |

**Difference between ArrayList and LinkedList**

| **Property** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| **Implementation** | ArrayList is implementation of List interface. | LinkedList internally uses a List & Queue interface |
|  | It’s internally use Array to store the elements | LinkedList internally uses a List & Doubly linked list to store the elements |
| **Declaration** | ArrayList<String> obj = new ArrayList<String>(); | LinkedList<String> obj = new LinkedList<String>(); |
| **Speed** | Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| **Best** | ArrayList is better for storing and accessing data | LinkedList is better for manipulating data. |
| **Act** | An ArrayList class can act as a list only because it implements List only. | LinkedList class can act as a list and queue both because it implements List and Deque interfaces |
| **Thread Safe** | ArrayList is not synchronized(so not thread safe) | LinkedList is not synchronized |
|  |  |  |
|  |  |  |

**Difference between LinkedList and LinkedHashSet**

| **Property** | **LinkedList** | **LinkedHashSet** |
| --- | --- | --- |
| **Implementation** | it’s implementation of List and deque interface. | it’s implementation of Set interface and it inherits Hashset class |
| **Internal implementation** | it’s internally implements uses doubly linked list to store the elements | it’s internally uses LinkedHashMap to store it’s elements. |
| **Declaration** | LinkedList<String> obj = new LinkedList<String>(); | LinkedHashSet<String> obj = new LinkedHashSet<String>(); |
| **Duplicates** | LinkedList could contains multiple number of duplicates in its collection. | LinkedHashSet contains unique elements only like HashSet. |
| **Null** | Any number of null value can be inserted in LinkedList without any restriction. | LinkedHashset allows only one null value in its collection |
| **compare the elements** | LinkedList use equals() method to compare the elements | LinkedHashSet also uses equals() and hashCode() methods to compare the elements |

**Difference between HashSet and LinkedHashSet**

| **Property** | **HashSet** | **LinkedHashSet** |
| --- | --- | --- |
| **Implements** | Set interface | Set interface |
| **Extends** | AbstractSet class | HashSet class |
| **Declaration** | HashSet<String> obj = new HashSet<String>(); | LinkedHashSet<String> obj = new LinkedHashSet<String>(); |
| **structure** | It uses a Hashtable to store the elements. | It uses doubly linked list to store and maintain the insertion order of the elements. |
| **Technique** | Hashing | Hashing |
| **Order** | It does not provide any insertion order. | It provides an insertion order; we can predict the order of elements. |
| **Null** | It allows only one null element. | It also allows only one null element. |
| **Synchronized** | Non-synchronized | Non-synchronized |
|  | It is recommended to use HashSet if you want a collection of unique elements and order of elements is not so important | LinkedHashSet is an ordered version of HashSet and it's maintains insertion order of the elements.  It is recommended to use if you want a collection of unique elements and order of elements is important |

**Difference between HashSet and HashMap**

| **Property** | **HashSet** | **HashMap** |
| --- | --- | --- |
| **Implements** | Set interface | Map interface |
| **Declaration** | HashSet<String> obj = new HashSet<String>(); | HashMap<String, Integer> obj = new HashMap<String, Integer>(); |
| **Duplicates** | Not allowed | Yes duplicates values are allowed but no duplicate key is allowed |
| **Null** | Have a single null value | Single null key and any number of null values |
| **Used** | HashSet we used to store objects(elements or values) | HashMap is used for storing key & value pairs |
| **When Used** | Always prefer when we do not maintain the uniqueness. | It is used when we need to maintain the uniqueness of data. |

**Difference between HashMap and HashTable**

| **Property** | **HashMap** | **HashTable** |
| --- | --- | --- |
| **Implements** | Map interface | Map interface |
| **Extends** | HashMap inherits AbstractMap class. | Hashtable inherits Dictionary class. |
| **Declaration** | HashMap<String, Integer> obj = new HashMap<String, Integer>(); | Hashtable<Integer,String> ht=new Hashtable<Integer,String>(); |
| **Synchronized** | HashMap is non synchronized. It is not-thread safe | Hashtable is synchronized. It is thread-safe and can be shared with many threads. |
| **Null** | HashMap allows one null key and multiple null values. | Hashtable doesn't allow any null key or value. |
|  |  |  |