**Arrays in Java –**

Arrays are first class objects in Java.

**public** **static** **void** main(String[] args) **throws** InterruptedException {

**int**[] arr = {1,2}; // 1 way to define Arrays

**for**(**int** x: arr) {

System.***out***.println(x);

}

// Output is 1 and 2

**int**[] arr1 = **new** **int**[2]; // Another way to define Arrays

**for**(**int** y:arr1) {

System.***out***.println(y);

}

// Output is 0 and 0

String[] str = **new** String[2];

**for**(String s:str) {

System.***out***.println(s);

}

// Output is null, null

**boolean** bnum[] = **new** **boolean**[5];

**for** (**boolean** val : bnum) // we can see here, even curly brace bracket is not even req

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

System.***out***.println();

// Output is false false ...

// COMPARISON OF 2 ARRAYS – ARRAY REFERENCE COMPARISON & ARRAY CONTENT COMPARISON --

**int** arr3[] = {1, 2, 3};

**int** arr4[] = {1, 2, 3};

**if** (arr3 == arr4) // Same as arr1.equals(arr2)

System.***out***.println("Same");

**else**

System.***out***.println("Not same");

// Output is Not same ... == and .equals method matches reference object

// Below method is to validate the contents of Arrays

Assert.*assertTrue*(Arrays.*equals*(arr3, arr4));

**if**(Arrays.*equals*(arr3, arr4)) {

System.***out***.println("Array contents are same"); //Output -Array contents are same

} }

Final Arrays in Java –

Logic is that if we declare here array as final, then we can't change its reference object. But we can change its values. So here we are trying to change the elements of the array which will be changed without any prob.

**final** **int**[] arr = {1,2,3};

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

{

arr[i] = arr[i]\*10;

System.***out***.println(arr[i]);

}

Output : 10 20 30

Now, we’ll create 1 more array and will try to change the reference object from 1st array to 2nd array.

**final** **int**[] arr = {1,2,3};

**int**[] arr2 = {3,4,5};

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

{

arr[i] = arr[i]\*10;

System.***out***.println(arr[i]);

}

arr = arr2; // We’ll get compilation error here itself. We can’t do this. If we want to do this, we have to remove final modifier from arr.

Once we do that, arr will get all the values of arr2, whether size of both array are same or not.

**Arrays Class in Java –**

The Arrays class in java.util package is a part of the Java Collection Framework. This class provides static methods to dynamically create and access Java arrays. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.

**Methods of Arrays Class** –

1. **Arrays asList() method** - The asList() method of java.util.Arrays class is used to return a fixed-size list backed by the specified array. Return Value: This method returns a list view of the specified array.

// Converting String array to List<String>

String [] arr1 = **new** String[] {"Test","Data"};

List<String> li = Arrays.*asList*(arr1);

System.***out***.println(li);

// Converting Integer array to List<Integer>

Integer[] arr2 = **new** Integer[] {3,4,5};

List<Integer> l2 = Arrays.*asList*(arr2);

System.***out***.println(l2);

Output : [Test, Data]

[3, 4, 5]

1. **Some more methods of Arrays Class – Will illustrate by below examples:**

// binarySearch method is used to search the exact location/index of given input in the given array

String [] arr1 = **new** String[] {"Test","Data","Charge"};

System.***out***.println(Arrays.*binarySearch*(arr1, "Test")); // Output is 2

Integer[] arr2 = **new** Integer[] {3,5,7,9};

System.***out***.println(Arrays.*binarySearch*(arr2, 5)); // output is 1

//Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order.

// Note : two array references are considered equal if both are null.

Integer[] arr3 = **new** Integer[] {3,5,7,9};

Integer[] arr4 = **new** Integer[] {3,5,9,7};

System.***out***.println(Arrays.*equals*(arr2, arr3)); // true - content & Order both are same

System.***out***.println(Arrays.*equals*(arr2, arr4)); // false - content is same but order is different

// sort - we can sort integers or String array as well

Arrays.*sort*(arr4);

System.***out***.println(Arrays.*toString*(arr4));

//We can sort in reverse order as well

Arrays.*sort*(arr4,Collections.*reverseOrder*());

System.***out***.println(Arrays.*toString*(arr4));

// sort an sub array

Integer[] arr5 = **new** Integer[] {3,53,7,19,22,90,01};

Arrays.*sort*(arr5,1,4);

System.***out***.println(Arrays.*toString*(arr5)); // So only elements from 1,2,3 index will be sort. (53,7 and 19 will sort)

// Output is [3, 7, 19, 53, 22, 90, 1]

// sorting an String array

Arrays.*sort*(arr1);

System.***out***.println(Arrays.*toString*(arr1)); // Output - [Charge, Data, Test]

Java Programs on Arrays :

1. **Java Program – Bubble Sort**

In this technique, iteration happen in multiple rounds and the highest element always reach to the right end in each iteration.

J loop will be used to do this activity, for adjacent comparison of items.

So, logic of sorting will apply mainly for j loop items :

Scenario 1 : If want to compare the integer number from Array, then we are going to use below program :

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

**int** [] a = {36,19,29,12,5};

**int** count =0; // This will count the no of times inner loop runs

**for**(**int** i=0;i<a.length-1;i++) {

**for**(**int** j=0;j<a.length-1-i;j++) {

count++;

**if**(a[j]>a[j+1]) {

**int** temp = a[j+1];

a[j+1] = a[j];

a[j] = temp;

}

}

// Below for loop is just purely for printing purpose of array after each iteration. Nothing else

**for**(**int** x:a) {

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

}

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

}

System.***out***.println(count); // 10

}

Scenario 2: Bubble sort for String:

We have to use compareTo() method to do that – compareTo() method is used to compare 2 String lexicographically. Each character of String is converted to a Unicode value of comparison. If both the string are equal then this method return 0 else it returns positive or negative value. The result is positive if first string is lexicographically greater than the 2nd String else result wud be negative.

Changes are only :

String [] a = {"Deepak","Bunny","Akash","Sam","Circle","peacock"};

String temp

**if**(a[j].compareTo(a[j+1])>0) {

We have to do only just these 3 changes.

1. **Java Program – Selection Sort**

**Ans**: Logic is , its an combination of Searching and sorting. It basically search the smallest element as compare to the first element, and once found it. It changes the position of the elements.

Code :

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

**int** arr[]={20,15,40,5,119,30};

**for**(**int** i=0;i<arr.length;i++) { // 1

**for**(**int** j=i+1;j<arr.length;j++) { // 2,3,4

**if**(arr[i]>arr[j]) {

**int** temp = arr[j];

arr[j]= arr[i];

arr[i] = temp;

}

}

**for**(**int** a : arr) {

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

}

System.***out***.println();

}

}

1. Java Program -
2. **How to find missing number in a Sorted Array in Java –**

Solution 1 : Linear Search

**(Sorted Array + one missing number + no duplicates present)**

**int**[] arr = {0,1,2,3,4,6,7,8,9};

**int** number = 0;

**for**(**int** i=0;i<arr.length;i++) { // n+1

**if**(i!=arr[i]) {

number = i;

**break**;

}

**else** **continue**;

}

System.***out***.println("Missing number: "+ number );

The above needs that array should start from 0 only. If its not, then it will give value 0 only.

**Solution 2 : Binary Search -**

**(Sorted Array + one missing number + no duplicates present)**

If we do that from linear search as mentioned above, its Time Complexity will be O(n) which is not good for a big no. of array. So, we have to follow some different approach which will be binary search.

Binary search concept works like, it will check the value at middle position index and if its less or greater than the middle position index, then it will take the remaining set of values and again the same process continues. In this way, in every iteration we are removing half of the values to traverse from the Loop.

We’ll use while loop for this activity, if we use for loop it will run for the given length of array. Its time complexity will be O(log n)

Code :

**public** **static** **void** main(String[] args) **throws** InterruptedException {

**int**[] arr1 = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18};

**int** missing = *intmissingNo*(arr1);

System.***out***.println(missing);

}

**public** **static** **int** intmissingNo(**int**[] arr) {

**int** left = 0;

**int** right = arr.length-1; // 17 initially

**while**(left<=right) {

**int** middle = (left+right) /2; // middle =8

**if**(arr[middle] != middle) { //

**if**(middle==0 || arr[middle-1]==middle-1) {

**return** middle;

}

right = middle-1;

}

**else** left = middle +1; // 1st lopp, left = 9 , right = 17, middle = 26/2 = 13 .. if arr[13]= 13 , left = 14, right = 17. middle = 31/2= 15

// left = 16. right = 17, middle = 16. // then left = 17, right = 17, middle = 17

}

**throw** **new** RuntimeException("no missing no");

}

}

**Solution 3 :** **If we suppose we know that we have an array which don’t have any duplicate no, and values starts from 1 only. It should definitely have 1, then we have 1 more solution to do that. Sum of numbers :**

Code :

**int**[] arr = {7,5,6,4,2,1,8,9,3};

**int** currentSum = 0;

**int** n = arr.length +1;

**int** sum = n\* (n+1)/2;

**for**(**int** i:arr) {

currentSum = currentSum + i;

}

**int** missingNumber = sum-currentSum;

System.***out***.println("Missing number= " + missingNumber ); // Output = 10

1. Sliding Window, Two Pointers, Fast and Slow Pointers, Merge Intervals, Cyclic Sort, and Top K elements that can help you to solve many frequently asked coding problems.
2. **Java Program – To find smallest and largest element in an array –**

There are 2 solutions for it, 1 is using Arrays.sort() method which has Time complexity of O(n logn)

Other is using selection sort, just sort the array and then we’ll get the smallest and largest integer. Its time complexity will be O(n).

Preferable is 1st solution

Arrays.sort()

**int**[] arr = {9,2,7,1,9,5,11,45,16};

Arrays.*sort*(arr); // O(n log n)

System.***out***.println("Smallest element is " +arr[0]);

System.***out***.println("largest element is "+ arr[arr.length-1]);

Output :

Smallest element is 1

largest element is 45

Using Selection Sort method –

**int**[] arr = {9,2,7,1,9,5,11,45,16};

**for**(**int** i=0;i<arr.length-1;i++) { // O(n)

**if**(arr[i]>arr[i+1]) {

**int** temp = arr[i+1];

arr[i+1] = arr[i];

arr[i] = temp;

}

}

**for**(**int** i:arr) {

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

}

System.***out***.println("Smallest element is " +arr[0]);

System.***out***.println("largest element is "+ arr[arr.length-1]);

Output - Same as from 1st Solution

1. **IMP Java Program - Find the number occurring odd number of times in an array –**

Explanation – Suppose there are many integers in an array, and we see that those are repeating. So basically we have to find that number which is repeating odd no. times.

Solution – We can do this with 2 for loops (Brute force method) but that will have time complexity of o(n^2)

Optimum solution will be by using Hashmap – It will have Time complexity of O(n)

Code :

**int** arr[] = **new** **int**[]{20, 40, 50, 40, 50, 20, 30, 30, 50, 20, 40, 40, 20};

HashMap<Integer, Integer> hs = **new** HashMap<Integer, Integer>();

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

**if**(hs.containsKey(arr[i])) {

hs.put(arr[i], hs.get(arr[i])+1);

}

**else** hs.put(arr[i], 1);

}

System.***out***.println(hs);

**for**(Map.Entry<Integer, Integer> m:hs.entrySet()) {

**if**(m.getValue()%2!=0) {

System.***out***.println("Odd element is "+m.getKey());

}

}

// Time Complexity is O(n)

Output –

{50=3, 20=4, 40=4, 30=2}

Odd element is 50

In above program, even it will give value for those array elements which are occurring only once.

1. **V IMP Java Program - Largest Sum Contiguous Array / KADANE’S ALGORITHM / Maximum Sum of Sub Array**

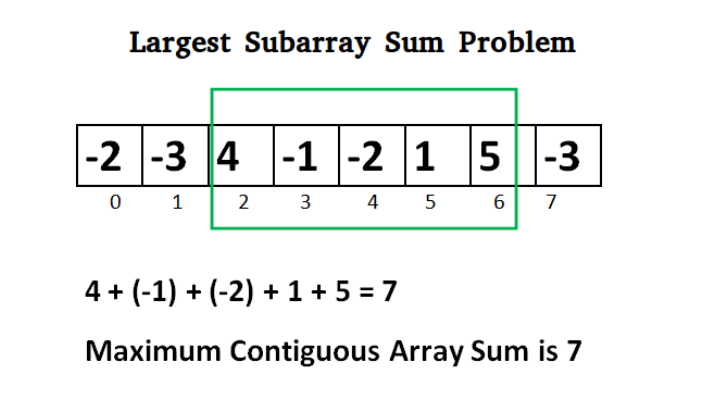
Code need is : We have an single dimensional array with no. of elements in it. We have to find the largest sum of a largest sub array

This Program is also known as **“KADANE’S ALGORITHM”**

We need to note 1 thing, there is no straight forward solution for this problem which can work for any type of input combo.

We will have diff-2 Code for the Array with only Negative integers, or Array with combination of + and – Integers.

**// Case 1 - Below Code is for Combination of + and – Integers.**



Code logic is we’ll define two int variables,

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

// Kadane's Algo --

**int** [] arr = {-2, -3, 4, -1, -2,-4,1, 5, -3,7};

**int** tempSum = 0;

**int** permSum = 0;

**int** firstIndex = 0;

**int** lastIdex = 0;

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

tempSum = tempSum + arr[i];

firstIndex+=1;

**if**(tempSum<0) {

tempSum = 0;

firstIndex=0;

}

**else** **if**(tempSum>permSum) {

permSum = tempSum;

lastIdex = i;

}

}

System.***out***.println("Max sum of a Substring is: "+permSum);

System.***out***.println("First Index will be: "+ (lastIdex-firstIndex+1));

System.***out***.println("Last Index is: "+lastIdex); // 9

Output :

Max sum of a Substring is: 10

FirstIndex will be: 6

Last Index is: 9

**// Case 2 - Below Code is for only negative integers –**

**Basically logic will be, if there are all negative numbers, then definitely the biggest number alone will maintain the sub array of maximum Sum.**

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

// Kadane's Algo --

**int** [] arr = {-31,-64, -22, -11,-55, -14, -114, -91};

**int** tempSum = arr[0];

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

**if**(tempSum>arr[i]) {

}

**else** tempSum = arr[i];

}

System.***out***.println("Max sum of a Substring is: "+tempSum);

**// Case 3 – FINAL SOLUTION-BELOW CODE WILL WORK FOR EVERY TYPE OF INPUT**

// int [] a = {-31,-64, -22, -11,-55, -14, -114, -5,-91};

//int [] a = {-2, -4, 6, -1, -3, 9,-6,2, 7, -4,8};

**int** [] a = {3,4,7,-8,1,9};

**int** sum=0;

**int** max=0;

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

sum=sum+a[i];

**if**(sum > max){

max=sum;

}

**if**(sum<=0){

sum=0;

}

}

System.***out***.println(max);

1. **IMP Java Program – Need to find sum of biggest 4 element contiguous array in a given Array –**

Program Need is - We have an array of say 6 or 7 elements, then we have to find the sum of that substring of size 4.

Code:

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

**int** [] arr = {1, 4, 3, 2,3, 5};

**int** temp=0;

**int** perm = 0;

**int** K = 4;

**for**(**int** i=0;i<=arr.length-4;i++) { // 0 1 2

**for**(**int** j=i;j<K+i;j++) { // 0 1 2 3

temp = temp+arr[j];

}

**if**(temp>perm) {

perm = temp;}

temp = 0;

}

System.***out***.println(perm);

}

1. **Java Program – In Addition of above program only, above we have find the max sum of the substring of 4 elements length. Now in this program we have to find the average of substring of 5 character length.**

Code :

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

**double** [] arr = {1, 3, 2, 6, -1, 4, 1, 8, 2};

**double** tempMaxSum = 0;

**double** avgSum = 0;

ArrayList<Double> it = **new** ArrayList<Double>();

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

**if**(arr.length-i >=5) {

**for**(**int** j=i;j<5+i;j++) {

tempMaxSum += arr[j];

}

avgSum = tempMaxSum/5;

it.add(avgSum);

tempMaxSum = 0;

}

}

System.***out***.println(it);

}

Imp – Time Complexity of above program is O(N \* K) -- K is the max length of substring

Output - [2.2, 2.8, 2.4, 3.6, 2.8]

1. **Java Program – We’ll do the same program - 10 in different way which will give the Time complexity as O(n) .**

Code **: Its an sliding window program**. We have to delete 1 char and add 1 char .

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

// Sliding window program

**double** [] arr = {1, 3, 2, 6,-1, 4, 1, 8, 2};

**double** tempMaxSum = 0;

**double** avgSum = 0;

ArrayList<Double> it = **new** ArrayList<Double>();

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

**if**(i<5) {

tempMaxSum = tempMaxSum + arr[i];

}

**else** {

avgSum = tempMaxSum/5;

it.add(avgSum);

tempMaxSum = tempMaxSum - arr[i-5] + arr[i+0];

}

}

avgSum = tempMaxSum/5;

it.add(avgSum);

System.***out***.println(it);

}

// Time Complexity is O(n)

1. **Java Program – Suppose we have Integer array and it has some duplicate elements as well. So we have to find a unique element output in ascending order**

**Ans** : We can use Set to do this activity.

**int** arr[] = {8,1,2,3,15,5,1,7,9,3};

Set<Integer> set = **new** HashSet<Integer>();

**for**(**int** a:arr) {

set.add(a);

}

System.***out***.println(set); // It will give only unique numbers and automatically in sorting order

Now, we get Set of unique numbers. But our aim was to remove duplicate elements from the array and give the ascending order array of unique elements. So we have to convert Set into array.

We can’t modify our original array (arr) in above case. We have to create a new array of the same size of Set and put all Set elements into newly created Array.

// Converting Set to Array --

Integer[] myArray = **new** Integer[set.size()];

set.toArray(myArray);

**for**(**int** uniqueElements : myArray) {

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

}

Explanation – Set object provides a method “toArray()” This method is to convert Set into Array Object.

Array.set() method in Java –

We have inbuilt method in Java, which is used to set a specified value to a specified index of a given Object array.

Syntax is :

Array.set(Object []array, int index, Object value)

We can use this method on Array Objects and Array primitive types as well.

// We have defined an Integer type array object below, and then we can set or change any value from any index

Integer[] array1 = **new** Integer[] {1,2,6,8,3};

Array.*set*(array1, 3, 112);

System.***out***.println("-----");

**for**(**int** uo : array1) {

System.***out***.println(uo);

}

System.***out***.println("///////%%%");

// We can set/change the value of int type array as well

**int**[] tush = **new** **int**[]{2,3,4};

Array.*set*(tush, 1, 98);

**for**(**int** up : tush) {

System.***out***.println(up);

}

// We can change or set a new value for String array as well

String[] tushar = **new** String[]{"test","data","charge"};

Array.*set*(tushar, 1, "numerono");

**for**(String work : tushar) {

System.***out***.println(work);

}

// ArrayIndexOutofbound exception will come if we try to set a value for any array which is not upto that length/index.

One more exception is java.lang.IllegalArgumentException:

This will come if we try to use this method on String not an array.

**Array.get() method** – This method is used to get the value at the given index.

**int** a[] = { 1, 2, 3, 4, 5 };

System.***out***.println(Array.*get*(a, 2));

**int** value = Array.*getInt*(a, 3);

System.***out***.println(value);

**char** b[] = {'G','f','G'};

System.***out***.println(Array.*getChar*(b, 2));

}

// This is the get() method, now there are other methods like getChar(), getFloat() but that will also give the same response as get() method gives.

Converting Array into List or Set –

Integer a[] = { 1, 2, 3, 4, 5 };

List<Integer> li = Arrays.*asList*(a); //converting array into list

// Now we’ll convert this List into Set .

// 1st way is to just traversing the list and printing the value. But we have 1 inbuilt method as well, which we'll see in 2nd case

Set<Integer> set = **new** HashSet<Integer>();

**for**(**int** ia : li) {

set.add(ia);

}

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

Set<Integer> set1 = **new** HashSet<Integer>(li);

System.***out***.println(set1);

1. **JAVA Program – To find the Longest consecutive subsequence in Array**

Suppose we have 1 array like int arr[] = {2, 6, 1, 9, 4,12,13, 5,3,11,10,15,16,14};

Then, we have to find the consecutive subsequence which can be in any place in the array but sequence should be maintained.

Code will be :

**int** arr[] = {2, 6, 1, 9, 4,12,13, 5,3,11,10,15,16,14};

**int** TempCount = 1;

**int** permCount = 0;

Arrays.*sort*(arr);

**for**(**int** i=0;i<arr.length-1;i++) {

**if**(arr[i+1]-arr[i]==1) {

TempCount = TempCount + 1;

**if**(i==arr.length-2) {

**if**(TempCount>permCount) {

permCount = TempCount;

}

}

}

**else** {

**if**(TempCount>permCount) {

permCount = TempCount;

}

TempCount = 1;

}

}

System.***out***.println("Longest seq is: "+permCount);

Other way of doing the same program is :

Don’t put that condition in else loop, in else loop just make the tempCount = 1, and keep that condition outside of if and else loop

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

**int** a[] = {2, 6, 17,1, 9, 4,12,13, 5,3,11,10,15,16,14};

**int** totalMaxLength = 1;

**int** maxSoFar = 1;

Arrays.*sort*(a);

**for**(**int** i = 1; i < a.length; i++) {

**if**(a[i] == a[i-1] + 1) {

maxSoFar++;

} **else** {

maxSoFar = 1;

}

**if**(maxSoFar > totalMaxLength) {

totalMaxLength = maxSoFar;

}

}

System.***out***.println(totalMaxLength);

}

1. **Java Program – 0-1 Knapsack Problem –**

Ans: Suppose we know that we have N items – say 4 items

Now we have 1 Integer array available which has value like “**int** [] val1 = {60,110,120,150};”

And weight like **int** [] wt = {10,20,30,40};

Now, we have given the max capacity of Knapsack as W = 70

Problem statement is – We have to find the maximum value of collection of weights which will be less than or equal to Kanpsack weight.

**Scenario 1 – Where we can’t break any item means, we have to take the complete weight of any item, we cant break it into sub weights.**

Logic Implementation will be – We’ll use Greedy algo to solve this problem. We’ll find that item whose value to weight ratio is max. Which means that our target is to get the max value, so find that item whose weight value is max.

Code: -

**int** [] wt = {10,20,30,40};

**float** [] val = {60,110,120,150};

**int** [] val1 = {60,110,120,150};

**int** knapWeight = 70;

HashMap<Integer, Float> hs = **new** HashMap<Integer, Float>();

HashMap<Integer, Integer> hs1 = **new** HashMap<Integer, Integer>();

// Putting all the data of values and weights in

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

hs.put(wt[i], (**float**) (val[i]/wt[i]))

hs1.put(wt[i], val1[i]);

}

Set<Map.Entry<Integer, Float>> set = hs.entrySet();

List<Float> ls = **new** ArrayList<Float>();

**for**(Map.Entry<Integer, Float> map : set)

{

ls.add(map.getValue());

}

Collections.*sort*(ls,Collections.*reverseOrder*());

LinkedHashMap<Integer, Float> newMap = **new** LinkedHashMap<Integer, Float>();

**for**(**int** i=0;i<ls.size();i++)

{

**for**(Entry<Integer, Float> map : set)

{

**if** (ls.get(i).equals(map.getValue()))

{

newMap.put(map.getKey(), map.getValue());

}

}

}

System.***out***.println(newMap);

**int** sum = 0;

**float** finalValue = 0;

**for**(Map.Entry<Integer, Float> map1 :newMap.entrySet()) {

sum = sum + map1.getKey();

**if**(sum<=knapWeight) {

finalValue = finalValue + hs1.get(map1.getKey());

}

/\* else {

float rem = knapWeight- (sum - map1.getKey());

float tempValue = (map1.getValue() \* rem);

finalValue = finalValue + tempValue;

break;

}\*/

}

System.***out***.println(finalValue);

**Scenario 2 – When we can break items and get the max value even in the decimals as well.**

So, in this case, above code in green color, which is commented out, just un-comment it. This is the code for the Scenario 2.

**Scenario 3 – Now, we have used Greedy algo to solve above 2 Scenarios, now we’ll use Dynamic Programming to solve Scenario 1.**

Logic is :

NOTE – Dynamic Programming is used to solve the optimization problems.

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

**int** val[] = {1,2,5,6};

**int** wt[] = {2,3,4,5};

**int** W=8,n=3;

**int** i, w;

**int** K[][] = **new** **int**[n + 1][W + 1];

**for** (i = 0; i <= n; i++) {

**for** (w = 0; w <= W; w++) {

**if** (i == 0 || w == 0)

K[i][w] = 0;

**else** **if** (wt[i - 1] <= w)

K[i][w] = *max*(

val[i - 1] + K[i - 1][w - wt[i - 1]],

K[i - 1][w]);

**else**

K[i][w] = K[i - 1][w];

}

}

System.***out***.println(K[n][W]);

}

**static** **int** max(**int** a, **int** b)

{

**return** (a > b) ? a : b;

}

}

1. **JAVA Program to put data into Json Files –**

We have to use JSONObject Class for these actions. JSON.Simple is the library that we need to import in pom.

Write Data into JSON Files –

1. JSONArray – If we want to write data in Json Arrays, we have to use add() method of JSONArray class
2. JSONObject - If we want to write data in Json Object ie. If we want to create just Json Objects, we have to use put() method of JSONObject class.

**Case 1 – When we** have suppose 3 pair of attributes, and we just want to store these 3 attributes directly in the Json file, then we have to follow below technique –

JSONObject employeDetails = **new** JSONObject();

employeDetails.put("Test1", "Tushar");

employeDetails.put("Test2", "Mittal");

employeDetails.put("Test3", "AVinash");

// Write this data into dumyfile.Json file --

**try** {

FileWriter file = **new** FileWriter("dummyfile.json");

file.write(employeDetails.toJSONString());

file.flush();

} **catch** (Exception e) {

e.printStackTrace();

}

In this way we are just creating 3 attributes in Json file and inserting the data.

Output will be :

{

"Test1":"Tushar",

"Test3":"AVinash",

"Test2":"Mittal"

}

**Case 2 – Now in the** above scenario, we want to store these 3 attributes in 1 Object, just an Object.

Then, Code will be :

JSONObject employeDetails = **new** JSONObject();

employeDetails.put("Test1", "Tushar");

employeDetails.put("Test2", "Mittal");

employeDetails.put("Test3", "AVinash");

JSONObject employeeObjec = **new** JSONObject();

employeeObjec.put("employee", employeDetails);

**try** {

FileWriter file = **new** FileWriter("dummyfile.json");

file.write(employeeObjec.toJSONString());

file.flush();

} **catch** (Exception e) {

e.printStackTrace();

}

Output will be :

{

"employee":{

"Test1":"Tushar",

"Test3":"AVinash",

"Test2":"Mittal"

}

}

SO, Object is employee, under which there are 3 attributes. Change in code of 2nd scenario will be just, we are storing values of 1st JSONObject into another JSONOBject.

**Case 3 – Now, we want** to create 2 separate Object, but not the arrays.

Code will be :

JSONObject employeDetails = **new** JSONObject();

employeDetails.put("Test1", "Tushar");

employeDetails.put("Test2", "Mittal");

employeDetails.put("Test3", "AVinash");

JSONObject employeeObjec = **new** JSONObject();

employeeObjec.put("Employee1", employeDetails);

employeeObjec.put("Employee2", employeDetails);

**try** {

FileWriter file = **new** FileWriter("dummyfile.json");

file.write(employeeObjec.toJSONString());

file.flush();

} **catch** (Exception e) {

e.printStackTrace();

}

Output will be:

{

"Employee2":{

"Test1":"Tushar",

"Test3":"AVinash",

"Test2":"Mittal"

},

"Employee1":{

"Test1":"Tushar",

"Test3":"AVinash",

"Test2":"Mittal"

}

}

Difference b/w Case 2 and Case 3 is just we have to create 1 more JSON Object and store the Attributes values into it.

Case 4 – IMP one, Now we have to store these Values in 1 JSON Array.

// 1st Object details

JSONObject employeDetails = **new** JSONObject();

employeDetails.put("Test1", "Tushar");

employeDetails.put("Test2", "Mittal");

employeDetails.put("Test3", "AVinash");

JSONObject employeeObjec = **new** JSONObject();

employeeObjec.put("Employee1", employeDetails);

// 2nd Employee Details

JSONObject employeDetails2 = **new** JSONObject();

employeDetails2.put("Test1", "Sam");

employeDetails2.put("Test2", "Peter");

employeDetails2.put("Test3", "John");

JSONObject employeeObjec2 = **new** JSONObject();

employeeObjec2.put("Employee2", employeDetails2);

// Storing above values in Json Array format

JSONArray employeeArray = **new** JSONArray();

employeeArray.add(employeeObjec); // Adding 1st object

employeeArray.add(employeeObjec2); // Adding 2nd object

// Write this data into Json file --

**try** {

FileWriter file = **new** FileWriter("dummyfile.json");

file.write(employeeArray.toJSONString());

file.flush();

} **catch** (Exception e) {

e.printStackTrace();

}

}

Output is :

[

{

"Employee1":{

"Test1":"Tushar",

"Test3":"AVinash",

"Test2":"Mittal"

}

},

{

"Employee2":{

"Test1":"Sam",

"Test3":"John",

"Test2":"Peter"

}

}

]

//////////// **API Testing / API Automation Testing ///////////**

Client-Server Architecture – This is a relation between 2 participants in information sharing, first one is who wants to consume specific info, called as Client in Client server architecture.

Other is Provider, who provides the info, called as Server in Client server architecture.

A Client and a Server establishes a connection using HTTP protocol. Once the connection is established, Client sends across the request to the Server in the form of XML or JSON which both entities (Client and Server) understand. After understanding the request Server responds with appropriate data by sending back a Response. The request and response means HTTP Request and Response.

**What is Rest - REST stands** for *Representation State Transfer*. REST is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet.

By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running.

REST Constraints –

1. Client -Server Architectural style - According to this constraint, an application should be modeled like a Client – Server. To relate to it, application should have the UI separate from the Data. There should be a distinct component handling Front end (UI) and the Back end (Database). UI here will act as a client and the back-end will be the server.

The importance of this style of design is that the Client and Server components can evolve independently.  Also, most important part is that a Server can serve multiple Clients, Clients can further be on different technologies.

Configuration of Rest Service to our Project – Just download Maven dependency of Rest Assured in pom file.

Rest API Test using Rest Assured –

Ques : ***What is Rest Assured ?***

Ans: We are going to use a popular tool for API Automation testing – Rest Assured. REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs. *Rest-Assured is a Java-based library that is used to test RESTful Web Services. This library behaves like a headless Client to access REST web services.* **Rest-Assured library also provides the ability to validate the HTTP Responses received from the server.**

We’ll use Http Method Type – GET.

**So in general process will be :**

Use the RestAssured class to generate a RequestSpecification for the URL( API Endpoint)

Specify the HTTP Method type

Send the Request to the Server

Get the Response back from the server

Print the returned Response’s Body

Code with Explanation –

RestAssured.*baseURI* = "https://reqres.in/api/products";

Explanation - This line uses a class called RestAssured to set up a request with the specified base URI. This class creates a http request that is going to use in further tests. It supports creating request of diff HTTP method types - GET, POST, PUT, PATH, DELETE, UPDATE, HEAD and OPTIONS.

It makes HTTP communication with the server and passes on the request that we create to get the details. It retrieves the response from the server.

RequestSpecification httpRequest = RestAssured.*given*();

Once the request is set up, store the Request in a variable so that it can be modified. This line basically allows to modify the request like adding headers or adding auth details.  Every Request in Rest-Assured library is represented by an interface called **RequestSpecification**.

Then, we have to set the content type of the request as json. So we’ll use below sentence .

httpRequest.header("Content-Type", "application/json");

Now, we have to call the server to get the resource ie, we have to issue a request to a server. And this task will be done by object of RequestSpecification class. We have request available of diff-2 HTTP Method types, like GET, POST etc. etc.

Response response = httpRequest.request(Method.***GET***, "/3");

Issuing request takes two arguments, first argument as HTTP Method Type and second as String (“/3”). This step actually sends the request to the remote server and gets a response back. This is why the return type of the request is specified as **Response**. Response Interface represents a Response returned from a server. This Response object will contain all the data sent by the server. We have various methods available for Response interface which is for diff-2 purposes that will see going further.

String responseBody = response.getBody().asString();

System.out.println(responseBody);

Now, above piece of code , getBody() method will give the body of the received response, which will convert into String using asString() method and we can print it easily.

Now, generally we see below things in Response :

**Response Status, Response Header and Response Body**

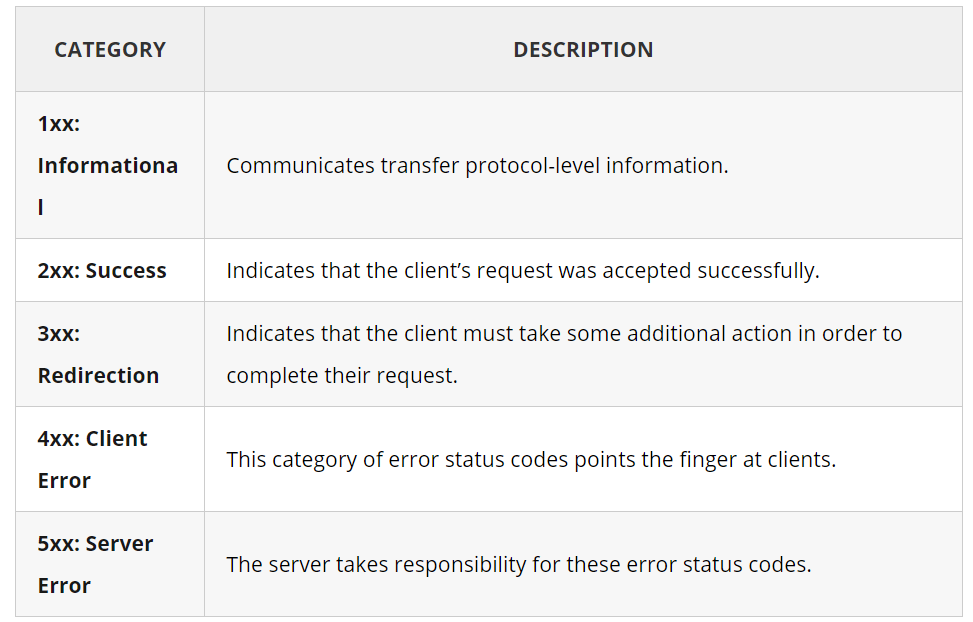
System.***out***.println(response.getStatusCode());

getStatusCode() method can be used to get the status code of the Response. It return an integer value.

**NOTE** : *Status codes* returned by the Server depends on whether the *Request* was successful or not. If the *Request* is successful, *Status Code 200* is returned. status code. If the Request is not successful, Status Code other than 200 will be returned. On high level, if there is any issue with Client side while sending a request, then generally 4xx series error comes, but if server is down then 5xx series error comes.

Please find below some imp Http error status codes -

**HTTP Error Status Codes :**



**Description for some main error codes -**

400 (Bad Request) - Errors can be like malformed request syntax, invalid request message parameters, or deceptive request routing etc.

401 (Unauthorized) - A 401 error response indicates that the client tried to operate on a protected resource without providing the proper authorization.

404(Not Found) - The 404 error status code indicates that the REST API can’t map the client’s URI to a resource but may be available in the future. Subsequent requests by the client are permissible.

500 (Internal Server Error) - 500 is the generic REST API error response. Most web frameworks automatically respond with this response status code whenever they execute some request handler code that raises an exception. Its not the client’s side error, mainly from Server side error.

504 (Gateway TimeOut Error) - Server error response.

System.***out***.println(response.getStatusLine());

The above method is used to get the stats line of the response. Output will be HTTP/1.1 200 OK. Explanation is First part is Http protocol **(HTTP/1.1).**Second is Status Code **(200). Third is the Status message (OK).**

**Headers -**

Every response that is received from a server contains zero or more headers. Headers are the part of Response that is sent by the server. Each header entry is basically a Key-Value pair. *One of the Headers called***Content-Type***which tells how to interpret the data present in the Body of the Response. If the Body contains data in the form of JSON, then the value of***Content-Type***header will be***application/json.***Similarly, if the data in the body is XML the***Content-Type***header will be***application/xml.**

System.***out***.println(response.getHeader("Content-Type"));

System.***out***.println(response.getHeader("Server"));

System.***out***.println(response.getHeader("Content-Encoding"));

Explanation – There are 2 methods, .header(String arg) and .getHeader(String arg). Both does the same thing, to get any particular individual header.

If we want to get all the Headers from the response. We’ll use .getHeaders() or .headers() method. Both will do the same job.

Headers allheaders = response.getHeaders();

**for**(Header header:allheaders ) {

System.***out***.println(header.getName() + "--- " + header.getValue());

}

OR

System.***out***.println(response.headers());

System.***out***.println(response.getHeaders());

**IMP Topic – Now we need to read the json response what we got from Server.**

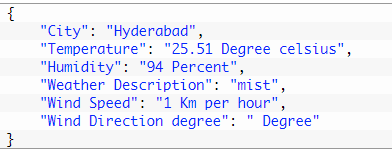
**Response**interface gives us a mechanism to extract nodes based on a given **JsonPath.** There is a method called **Response.JsonPath(),**which returns a **io.restassured.path.json.JsonPath**Object.  This object can be used to further query specific parts of the **Response Json**.

JsonPath jsonPathEvaluator = response.jsonPath();

We got the JsonPath object instance from the Response Interface.

One of the most important advantage of *JSON* is that it is a light weight format that can be used to interchange data between computers and processes. *JSON* , like XML, is a format to provide structure to the data.

Ex 1 : We go the below response, now we have to get the data based on our situation.



That’s the simple response, now we have to find the Json path to reach to the property name that we want. We want json path or value for “City”.

Json Path will be : $.City

But when we have to write this in Java, we can skip $ sign

So, Java JsonPath will be City

Code will be ;

String city = jsonPathEvaluator.get("City");

System.out.println("City received from Response " + city);

Output - Hyderabad

Example 2 : We need to find Property Value of company in “ad” container and name in “data” container.

{

   "data":{

      "id":3,  
      "name":"true red",  
      "year":2002,  
      "color":"#BF1932",  
      "pantone\_value":"19-1664"

},  
   "ad":{

      "company":"StatusCode Weekly",  
      "url":"http://statuscode.org/",  
      "text":"A weekly newsletter focusing on software development, infrastructure, the server, performance, and the stack end of things."

}

}

Code will be :

String company = jsonPathEvaluator.get("ad.company");

System.***out***.println(company);

String data = jsonPathEvaluator.get("data.name");

System.***out***.println(data);

Now, if we want to print the whole container, “ad”, then this one is a HashMap. Code will be

HashMap<String, String> books = jsonPathEvaluator.*from*(responseBody).get("ad"); // to get the whole Map value of ad array.

System.***out***.println(books);

Output :

{company=StatusCode Weekly, text=A weekly newsletter focusing on software development, infrastructure, the server, performance, and the stack end of things., url=http://statuscode.org/}

**POST Request -**

POST method is used to send data to the server. Common places where you can find a POST request is when you submit Form Data (HTML forms) on a web page.

The data that is sent to the server in a POST request is sent in the body of the HTTP request. The type of body, XML, JSON or some other format is defined by the Content-Type header. If a POST request contains JSON data then the Content-Type header will have a value of application/json. Similarly, for a POST request containing XML the Content-Type header value will be application/xml.

**IMP TOPIC – REASONS OF GETTING ERROR RESPONSE CODES-**

1. 201 – I just changed the HTTP request type from GET to POST, I got this error, since the API is expected the GET Request type for that particular API

Correct one :

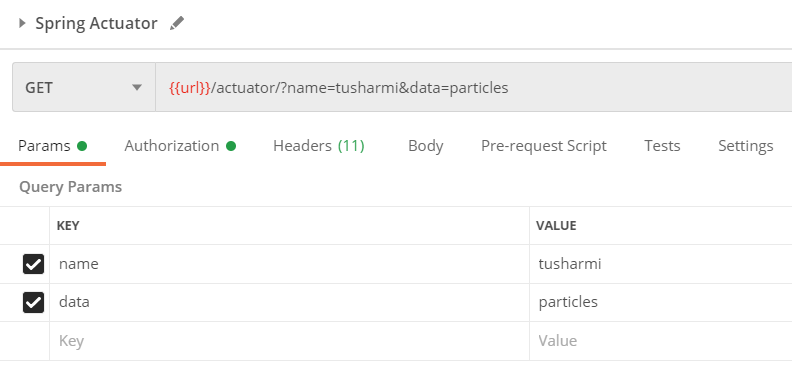
Response response = httpRequest.request(Method.***GET***, "/3");

Incorrect Changes done below :

Response response = httpRequest.request(Method.***POST***, "/3");

**Ques 1 – How to pass Query parameters to Rest Assured (GET Request) –**

Ans : Suppose we have Get http request to hit a end point, and we need to pass some parameters, which we say as “Params” in term of Postman.



We can see, User is trying to pass 2 parameters “name and data”. It has some values as well.

***Solution is : REST Assured will automatically try to determine which parameter type (i.e. query or form parameter) based on the HTTP method. In case of GET query parameters will automatically be used and in case of POST form parameters will be used.***

In above case, these are query parameters, so we need to pass like below :

Below solution is for single parameter -

RestAssured.*given*().contentType(ContentType.***JSON***).queryParam("name", "tusharmi");

Below solution is for multiple params –

RestAssured.*given*().contentType(ContentType.***JSON***).queryParam("name", "tusharmi").queryParam("data", "particles");

A set of parameters attached to the end of the URL is called **Query Parameters**. They are appended to the URL by adding ‘?’ at the end of the URL.

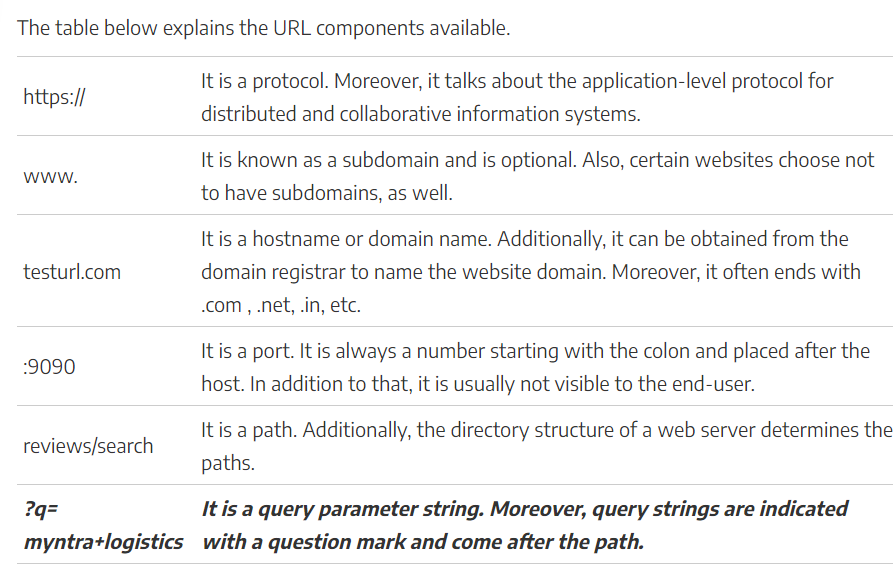
**Composition of URI –**

URI stands for URI (Uniform Resource Identifier).

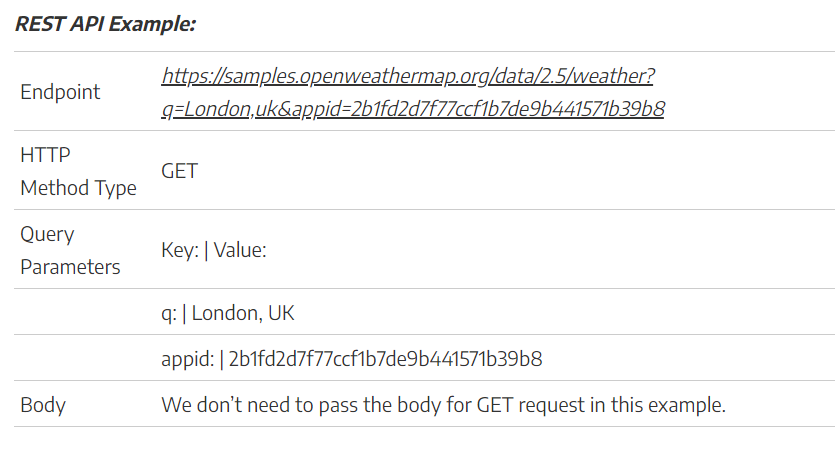
URI format is -

<protocol>://<service-name>/<ResourceType>/<ResourceID>

the URI is **http://localhost:8080/UserManagement/rest/UserService/users** and the VERB is GET.



Example is :



Now, we ‘ll write below code for Query parameters,

RestAssured.*baseURI* = "https://reqres.in/api/products";

RequestSpecification httpRequest = RestAssured.*given*();

Response response = httpRequest.contentType(ContentType.***JSON***).queryParam("name", "tushar").get("/method");

We can pass 2 query parameters as well like below –

RestAssured.*given*().contentType(ContentType.***JSON***).queryParam("name", "tusharmi").queryParam("data", "particles");

We can even use external sources to pass params data like HAshmap – Yet to try – but generally will pass this as body parameter. When we want to send data from Hashmap.

Ques 2 -How to pass Path parameters

Ans – In place of queryParam() method, we have to use pathPAram().

Remaining things will be same.

Get Operation and Assertions in its Response –

Then().body(“Author”, is(“Karthik KK”) //validation of Prperty tag – Author

Then().body(“autho”, containsInAnyOrder(“Karthik KK, null).statusCode(200)

So, generally statement written after then() is for the assertion statement

**Ques 3 – POST request with body parameter –**

Means, we see POST request type need a body , json body, then we can use Hashmap to do that.

Just create 1 Hashmpa , and pass that object in

With().body(hs).

HashMap<String, String> hs = **new** HashMap<String, String>();

RestAssured.*given*().contentType(ContentType.***JSON***).with().body(hs).when().get(RestAssured.*baseURI* )