Product and sum of number is same as original number

for(int i=10;i<=99;i++){

int k=i%10;

int m=i/10;

if(i==(k\*m)+(k+m)){

System.out.println(i);

Even or Odd without Mod

Scanner sc = new Scanner(System.in);

int n= sc.nextInt();

if((n/2)\*2==n){

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

}else{

System.out.println("Odd number");

       }

2nd method

Scanner sc = new Scanner(System.in);

int n= sc.nextInt();

if((n&1)==0){

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

}else{

System.out.println("Odd number");

       }

Swap of number using bitwise

Scanner sc = new Scanner(System.in);

int n= sc.nextInt();

int m = sc.nextInt();

n=n^m;

m=n^m;

n=n^m;

System.out.println(n);

System.out.println(m);

Amstrong number

public static void main(String[] args) {

int n = 153;

int count = 0;

int t = n;

while(t!=0){

t = t/10;

count ++;

}

int s =0;

t=n;

while(t!=0){

int d=t%10;

s+=Math.pow(d,count);

t=t/10;

}

if(s==n){

System.out.println("Amstrong number");

}else{

System.out.println("It is not a Amstrong number");

}

Kapraker Value

int n = 297;

int square = n \* n;

int t = square;

int digits = 0;

while (t > 0) {

digits++;

t /= 10;

}

int count = 1;

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

count = count \* 10;

int l = square / count;

int r = square % count;

if (r > 0 && (l + r == n)) {

System.out.println("It is a Kaprekar number");

}

}

Rotate array by one

public static void main(String[] args) {

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

int n= arr.length;

int last=arr[n-1];

for(int i=n-1;i>0;i--){

arr[i]=arr[i-1];

}

arr[0]=last;

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

System.out.println(arr[i] +" ");

}

}

}

rotate array by d  
int arr[] = {1, 2, 3, 4, 5};

int n = arr.length;

int last = arr[n - 1];

int d = 2;

int[] t= new int[5];

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

t[i] = arr[n - d + i];

}

for (int i = n-1; i>=d; i--) {

arr[i] = arr[i-d];

}

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

arr[i]=t[i];

}

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

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

}

Sending all zeros to last without sorting

int arr[]={1,0,2,0,3,5};

int[] t= new int[6];

int n=arr.length;

int d=6;

int count =0;

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

if(arr[i]!=0){

arr[count++]=arr[i];

}

}

while(count<n){

arr[count++] =0;

}

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

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

}

Sum of Max and Min value in an array

int arr[]={20,30,40,50,100};

int[] t= new int[6];

int n=arr.length;

int max=arr[0];

for (int i = 0; i<n-1; i++) {

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

max=arr[i+1];

}

}

int min=arr[0];

for (int i = 0; i<n-1; i++) {

if(arr[i]<min){

min=arr[i+1];

}

}

System.out.println(min+max);

Sliding Window

int arr[]={20,30,40,50,100};

int n=arr.length;

int sum =0;

int target=190;

int k=3;

if(n<k){

System.out.println("The n is less than k"+k);

return;

}

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

sum += arr[i];

}

if(sum==target){

System.out.println("target found");

}

for (int i = k; i<n; i++) {

sum+=arr[i]-arr[i-k];

if(sum==target){

System.out.println("target found ");

}

}

Zig-Zag Pattern

int n=3;

int col=13;

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

for(int j=1;j<col;j++){

if((i==1&&j%4==3)||(i==2&&j%2==0)||(i==3&&j%4==1)){

System.out.print("\* ");

}else{

System.out.print(" ");

}

}

System.out.println();

}

Fibo using direct recursion

public int fibo(int n){

if(n<=1){

return n;

}

return fibo(n-1)+fibo(n-2);

}

public static void main(String[] args) {

int num=11;

Main t = new Main();

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

System.out.print(t.fibo(i)+ " ");

}

}

Using indirect recurrison

{

int result=0;

int oddnum(int odd){

if(odd%2!=0){

return result=odd+1;

}else{

return evennum(odd);

}

}

int evennum(int even){

if(even%2==0){

return result=even-1;

}else{

return oddnum(even);

}

}

public static void main(String[] args) {

int num=10;

Main t = new Main();

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

System.out.print(t.oddnum(i)+ " ");

}

}

series of odd+1 and even-1;

Method overriding and overloading  
class Calculator{

int add(int a, int b){

return a + b;

}

double add( double a ,double b){

return a+b;

}

int add(int a ,int b,int c){

return a+b+c;

}

}

public class Main

{

public static void main(String[] args) {

Calculator calc= new Calculator();

int a=10;

int b=20;

double c=25.5;

System.out.println(calc.add(a,c));

}

}  
  
Sliding window with index and max sum

int arr[]={200,30,40,50,100};

int n=arr.length;

int sum =0;

int max=0;

int k=3;

int count=0;

if(n<k){

System.out.println("The n is less than k"+k);

return;

}

int eind=0;

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

count++;

eind=count;

sum += arr[i];

}

if(sum>max){

max=sum;

}

for (int i = k; i<n; i++) {

sum+=arr[i]-arr[i-k];

if(sum>max){

max=sum;

count++;

}

eind=count-1;

}

System.out.println(max);

System.out.println(eind-2);

System.out.println(eind);

Direct recursion

public int fibo(int n){

if(n<=1){

return n;

}

return fibo(n-1)+fibo(n-2);

}

public static void main(String[] args) {

int num=11;

Main t = new Main();

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

System.out.print(t.fibo(i)+ " ");

}

}

Indirect Recurrision

{

int result=0;

int oddnum(int odd){

if(odd%2!=0){

return result=odd+1;

}else{

return evennum(odd);

}

}

int evennum(int even){

if(even%2==0){

return result=even-1;

}else{

return oddnum(even);

}

}

public static void main(String[] args) {

int num=10;

Main t = new Main();

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

System.out.print(t.oddnum(i)+ " ");

}

}

series of odd+1 and even-1;  
  
  
parametarized constructor

class Person{

String name;

int age;

Person(){

this("Sachin", 40);

}

Person(String name, int age){

this.name = name;

this.age = age;

}

void display(){

System.out.println("Name "+name+" Age "+age);

}

}

public class Main{

public static void main(String[] args) {

Person t = new Person();

t.display();

}

}

default in parameterized constroctur  
  
  
single level Inheritance  
class Animal{

void makesound(){

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal{

void bark(){

System.out.println("Dog Barks");

}

}

public class Main{

public static void main(String[] args) {

Dog t = new Dog();

t.makesound();

t.bark();

}

}

Single Level Inheritance  
  
Hamming Weigtht  
int n=13;

int count=0;

while(n>0){

if((n&1)==1){

count++;

}

n >>=1;

}

System.out.println(count);

Hamming weight

find the maximum length of odd or even sequence number

int arr[]={4,8,6};

int n= arr.length;

int maxLength=1;

int count=1;

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

if((arr[i]%2==0 && arr[i-1]%2==1)||(arr[i]%2==1 && arr[i-1]%2==0)){

count++;

if(maxLength<count){

maxLength=count;

}

}else{

count=1;

}

}

System.out.println(maxLength);

Exception handling

static void checkNumber(int n) throws Exception{

if(n<0){

throw new Exception("Negetive number not allowed" );

}

System.out.println("Valid number:"+n);

}

public static void main(String[] args) {

try {

checkNumber(-5);

} catch(Exception e) {

System.out.println("Caught an Exception " +e.getMessage());

}finally{

System.out.println("Execution ended");

}

}

Docking

static void riskyMethod() throws ArithmeticException{

System.out.println("Inside RiskyMethod");

throw new ArithmeticException("Division by Zero Occured!");

}

static void callerMethod() throws ArithmeticException{

System.out.println("Inside CallerMethod");

riskyMethod();

}

public static void main(String[] args) {

try {

callerMethod();

} catch(ArithmeticException e) {

System.out.println("Caught an Exception " +e.getMessage());

}finally{

System.out.println("Execution ended");

}

}

Docking  
  
Wrapper Auto-Boxing

public static void main(String[] args) {

int num =10;

Integer obj=num;

double d= 5.5;

Double dobj=d;

char c='A';

Character cobj=c;

System.out.println("Primitive int "+ obj);

System.out.println("Primitive double "+ dobj);

System.out.println("Primitive Charcter "+ cobj);

}

Collection Classes

ArrayList

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

public class Main

{

public static void main(String[] args) {

ArrayList<Integer> list1= new ArrayList<Integer>();

list1.add(20);

list1.add(10);

list1.add(30);

list1.add(40);

list1.add(50);

Iterator<Integer> itr = list1.iterator();

while(itr.hasNext()){

System.out.println("VAlue :"+itr.next());

}

}

}

adding and displaying values in the ArrayList

ArrayLIst Operations

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

public class Main

{

public static void main(String[] args) {

ArrayList<Integer> list1= new ArrayList<Integer>();

int count =0;

list1.add(10);

list1.add(20);

list1.add(30);

list1.add(40);

list1.add(50);

list1.add(2,25);

System.out.println("Value present at the index 3 :"+list1.get(3));

System.out.println("the Value present or not :"+list1.contains(30));

list1.remove(3);

System.out.println("length of the linkedlist "+count);

System.out.println("Size of the list :"+list1.size());

System.out.println("Value present at the index 3 :"+list1.indexOf(40));

Iterator<Integer> itr = list1.iterator();

while(itr.hasNext()){

count++;

System.out.println("VAlue :"+itr.next());

}

}

}

Why we use the linked list instead of the Array List because it uses shifting it will get Error

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

public class Main

{

public static void main(String[] args) {

ArrayList<Integer> list1= new ArrayList<Integer>();

for(int i=1;i<=10;i++){

list1.add(i);

}

for(int i=0;i<list1.size();i++){

if(list1.get(i)>7){

list1.remove(i);

i--;

}

}

System.out.println("Number remaining "+list1);

}

}  
  
Adding the diff datatypes in single Arraylist

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

public class Main

{

public static void main(String[] args) {

ArrayList<Object> list1= new ArrayList<Object>();

list1.add(5);

list1.add(6.5);

list1.add(-1);

list1.add("Sachin");

System.out.println(list1);

Iterator<Object> itr = list1.iterator();

while(itr.hasNext()){

System.out.println("VAlue :"+itr.next());

}

}

}

ArrayList Single line initialization of values

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

public class Main

{

public static void main(String[] args) {

ArrayList<Object> list1= new ArrayList<>(Arrays.asList(10,20,30,40,50));

// list1.add(5);

// list1.add(6.5);

// list1.add(-1);

// list1.add("Sachin");

System.out.println(list1);

// Iterator<Object> itr = list1.iterator();

// while(itr.hasNext()){

// System.out.println("VAlue :"+itr.next());

// }

}

}

Arraylist sort and reverse using builtin function

import java.util.ArrayList;

import java.util.Iterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

ArrayList<Object> list1= new ArrayList<>(Arrays.asList(20,10,30,5,50));

Collections.reverse(list1);

System.out.println("reveserd Number "+list1);

Collections.sort(list1);

System.out.println(list1);

LinkedList Diff Operations

import java.util.LinkedList;

import java.util.Iterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

LinkedList<Integer> list1= new LinkedList<>();

list1.add(10);

list1.addFirst(20);

list1.addLast(50);

list1.getFirst();

System.out.println(list1.remove(1));

System.out.println(list1);

// Iterator<Object> itr = list1.iterator();

// while(itr.hasNext()){

// System.out.println("VAlue :"+itr.next());

// }

}

}

LinkedList with accending and decending order

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

LinkedList<Integer> list1= new LinkedList<>();

Scanner sc = new Scanner(System.in);

System.out.println("Enter 5 number:");

for(int i=0;i<5;i++){

list1.add(sc.nextInt());

}

System.out.println("Forword Traversal");

Iterator<Integer> itr = list1.iterator();

while(itr.hasNext()){

System.out.println("VAlue :"+itr.next());

}

System.out.println("Backword Traversal");

Iterator<Integer> itr2 = list1.descendingIterator();

while(itr2.hasNext()){

System.out.println("VAlue :"+itr2.next());

}

}

}

With using ListIterator for previous

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

LinkedList<Integer> list1= new LinkedList<>();

Scanner sc = new Scanner(System.in);

System.out.println("Enter 5 number:");

for(int i=0;i<5;i++){

list1.add(sc.nextInt());

}

System.out.println("Forword Traversal");

Iterator<Integer> itr = list1.iterator();

while(itr.hasNext()){

System.out.println("VAlue :"+itr.next());

}

System.out.println("Backword Traversal");

ListIterator<Integer> itr2 = list1.listIterator(list1.size());

while(itr2.hasPrevious()){

System.out.println("VAlue :"+itr2.previous());

}

}

}

Merging of tw0 linked list

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

LinkedList<Integer> stack= new LinkedList<>();

Scanner sc = new Scanner(System.in);

list1.add(10);

list1.add(20);

list1.add(30);

list2.add(40);

list2.add(50);

list2.add(60);

LinkedList<Integer> mergeList = new LinkedList<>(list1);

mergeList.addAll(list2);

System.out.println("the merged list "+mergeList);

mergeList.addAll(2,list2);

System.out.println("the merged from 2 index oflist "+mergeList);

}

}

Stack Operations

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

import java.util.\*;

import java.util.Collections;

public class Main

{

public static void main(String[] args) {

Stack<Integer> stack1= new Stack<>();

Scanner sc = new Scanner(System.in);

stack1.push(10);

stack1.push(20);

stack1.push(30);

stack1.push(40);

stack1.push(50);

stack1.push(60);

System.out.println("stack elements are"+stack1);

System.out.println("stack peek element "+stack1.peek());

System.out.println("stack poped element "+stack1.pop());

System.out.println("stack Searched element "+stack1.search(20));

System.out.println("stack is empty "+stack1.isEmpty());

System.out.println("stack elements are "+stack1);

}

}

HashSet Operations

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

import java.util.\*;

import java.util.Collections;

public class Main{

public static void main(String[] args) {

HashSet<Integer> hashset= new HashSet<>();

LinkedHashSet<Integer> linkedset= new LinkedHashSet<>();

TreeSet<Integer> treeset= new TreeSet<>();

int[] numbers={50,20,40,10,30,20,40,10,40,45,60,30,24};

for(int num : numbers){

hashset.add(num);

linkedset.add(num);

treeset.add(num);

}

System.out.println("hashset unorder set"+hashset);

System.out.println("linkedhashset Insertion order"+linkedset);

System.out.println("Treeset Sorted order"+treeset);

}

}