**1.public** **class** BubbleSortEx {

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

**int**[] a = **new** **int**[] { 11, 13, 5, 7, 3 };

System.***out***.print("Before : ");

*print*(a);

**for** (**int** outer = a.length - 1; outer > 0; outer--) {

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

**for** (**int** inner = 0; inner < outer; inner++) {

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

**int** temp = a[inner];

a[inner] = a[inner + 1];

a[inner + 1] = temp;

}

*print*(a);

}

}

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

System.***out***.print("After : ");

*print*(a);

}

**private** **static** **void** print(**int**[] a) {

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

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

}

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

}

}

/\*

Before : 11 13 5 7 3

-----

11 13 5 7 3

11 5 13 7 3

11 5 7 13 3

11 5 7 3 13

-----

5 11 7 3 13

5 7 11 3 13

5 7 3 11 13

-----

5 7 3 11 13

5 3 7 11 13

-----

3 5 7 11 13

=====

After : 3 5 7 11 13

\*/

**2. public** **class** BubbleSortOnString {

String sortMethod(String s) {

**char** ch[] = s.toCharArray();

**char** chTemp;

// bubble Sort - to perform reverse sorting

**for** (**int** i = ch.length - 1; i > 1; i--) {

**for** (**int** j = 0; j < i; j++) {

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

// Swap logic in below 3 lines

chTemp = ch[j];

ch[j] = ch[j + 1];

ch[j + 1] = chTemp;

}

}

}

**return** **new** String(ch);

}

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

BubbleSortOnString o = **new** BubbleSortOnString();

String stringToBeSorted = "java";

// String stringToBeSorted= "javaMadeSoEasy"; //Test String

System.***out***.println(o.sortMethod(stringToBeSorted));

}

}

/\*

\* OUTPUT

\*

\* aajv

\*

\*/

**3. public** **class** InsertionSortEx {

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

**int**[] a = **new** **int**[] { 11, 13, 5, 7, 3 };

System.***out***.print("Before : ");

*print*(a);

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

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

**for** (**int** inner = outer; inner > 0; inner--) {

**if** (a[inner] < a[inner - 1]) {

**int** temp = a[inner];

a[inner] = a[inner - 1];

a[inner - 1] = temp;

}

*print*(a);

}

}

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

System.***out***.print("After : ");

*print*(a);

}

**private** **static** **void** print(**int**[] a) {

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

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

}

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

}

}

/\*

\*

Before : 11 13 5 7 3

-----

11 13 5 7 3

-----

11 5 13 7 3

5 11 13 7 3

-----

5 11 7 13 3

5 7 11 13 3

5 7 11 13 3

-----

5 7 11 3 13

5 7 3 11 13

5 3 7 11 13

3 5 7 11 13

=====

After : 3 5 7 11 13

\*/

**4. public** **class** SelectionSortEx {

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

**int**[] a = **new** **int**[] { 11, 13, 5, 7, 3 };

System.***out***.print("Before : ");

*print*(a);

**int** n = a.length;

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

**int** mininum = outer;

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

**for** (**int** inner = outer + 1; inner < n; inner++)

**if** (a[inner] < a[mininum])

mininum = inner;

**int** temp = a[mininum];

a[mininum] = a[outer];

a[outer] = temp;

*print*(a);

}

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

System.***out***.print("After : ");

*print*(a);

}

**private** **static** **void** print(**int**[] a) {

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

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

}

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

}

}

/\*

\*

Before : 11 13 5 7 3

-----

3 13 5 7 11

-----

3 5 13 7 11

-----

3 5 7 13 11

-----

3 5 7 11 13

=====

After : 3 5 7 11 13

\*/

**5. public** **class** MergeSortEx {

**private** **int**[] array;

**private** **int**[] tempMergArr;

**private** **int** length;

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

**int**[] inputArr = { 11, 13, 5, 7, 3 };

MergeSortEx mms = **new** MergeSortEx();

mms.sort(inputArr);

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

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

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

}

}

**public** **void** sort(**int** inputArr[]) {

**this**.array = inputArr;

**this**.length = inputArr.length;

**this**.tempMergArr = **new** **int**[length];

doMergeSort(0, length - 1);

}

**private** **void** doMergeSort(**int** lowerIndex, **int** higherIndex) {

**if** (lowerIndex < higherIndex) {

**int** middle = lowerIndex + (higherIndex - lowerIndex) / 2;

// Below step sorts the left side of the array

doMergeSort(lowerIndex, middle);

// Below step sorts the right side of the array

doMergeSort(middle + 1, higherIndex);

// Now merge both sides

mergeParts(lowerIndex, middle, higherIndex);

}

}

**private** **void** mergeParts(**int** lowerIndex, **int** middle, **int** higherIndex) {

**for** (**int** i = lowerIndex; i <= higherIndex; i++) {

tempMergArr[i] = array[i];

}

**int** i = lowerIndex;

**int** j = middle + 1;

**int** k = lowerIndex;

**while** (i <= middle && j <= higherIndex) {

**if** (tempMergArr[i] <= tempMergArr[j]) {

array[k] = tempMergArr[i];

i++;

} **else** {

array[k] = tempMergArr[j];

j++;

}

k++;

}

**while** (i <= middle) {

array[k] = tempMergArr[i];

k++;

i++;

}

}

}

**6. public** **class** MergeSortEx1 {

**static** **int**[] *ar* = { 54, 13, 24, 19, 11, 3, 71, 8 };

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

System.***out***.print("Display array before Merge sorting: ");

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

System.***out***.print(*ar*[i] + " ");

**int**[] ar2 = **new** **int**[*ar*.length];

*mergeSort*(ar2, 0, *ar*.length - 1);

System.***out***.print("\nDisplay array after Merge sort: ");

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

System.***out***.print(*ar*[i] + " ");

}

**static** **void** mergeSort(**int**[] workSpace, **int** lowerBound,

**int** upperBound) {

**if** (lowerBound == upperBound) {

**return**; // no need to sort further.

} **else** { // finding mid point

**int** midPoint = (lowerBound + upperBound) / 2;

*mergeSort*(workSpace, lowerBound, midPoint); // sorting low half

*mergeSort*(workSpace, midPoint + 1, upperBound); // sort upper alf

*merging*(workSpace, lowerBound, midPoint + 1, upperBound);//

}

}

**static** **void** merging(**int**[] ar2, **int** low, **int** high, **int** upperBound) {

**int** midPoint = high - 1;

**int** lowerBound = low;

**int** n = upperBound - lowerBound + 1;

**int** i = 0;

**while** (low <= midPoint && high <= upperBound)

**if** (*ar*[low] < *ar*[high])

ar2[i++] = *ar*[low++];

**else**

ar2[i++] = *ar*[high++];

**while** (low <= midPoint)

ar2[i++] = *ar*[low++];

**while** (high <= upperBound)

ar2[i++] = *ar*[high++];

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

*ar*[lowerBound + i] = ar2[i];

}

}

/\*

\* OUTPUT

\*

\* Display array before Merge sorting: 54 13 24 19 11 3 71 8

\* Display array after Merge sort: 3 8 11 13 19 24 54 71

\*

\*/

**7. public** **class** LinearSearchEx {

**public** **static** **int** linerSearch(**int**[] arr, **int** key) {

**int** size = arr.length;

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

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

**return** i;

}

}

**return** -1;

}

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

**int**[] arr1 = { 23, 45, 21, 55, 234, 1, 34, 90 };

**int** searchKey = 34;

System.***out***.println("Key " + searchKey + " found at index: " + *linerSearch*(arr1, searchKey));

**int**[] arr2 = { 123, 445, 421, 595, 2134, 41, 304, 190 };

searchKey = 421;

System.***out***.println("Key " + searchKey + " found at index: " + *linerSearch*(arr2, searchKey));

}

}

**8. public** **class** BinarySearchEx {

**public** **int** binarySearch(**int**[] a, **int** key) {

**int** start = 0;

**int** end = a.length - 1;

**while** (start <= end) {

**int** mid = (start + end) / 2;

**if** (key == a[mid]) {

**return** mid;

}

**if** (key < a[mid]) {

end = mid - 1;

} **else** {

start = mid + 1;

}

}

**return** -1;

}

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

BinarySearchEx mbs = **new** BinarySearchEx();

**int**[] arr = { 2, 4, 6, 8, 10, 12, 14, 16 };

System.***out***.println("Key 14's position: " + mbs.binarySearch(arr, 14));

**int**[] arr1 = { 6, 34, 78, 123, 432, 900 };

System.***out*** .println("Key 432's position: " + mbs.binarySearch(arr1, 432));

}

}

**public** **class** StackEx {

**private** **int** size;

**private** **int**[] stack;

**private** **int** top;

**public** StackEx(**int** size) {

**this**.size = size;

**this**.stack = **new** **int**[size];

**this**.top = -1;

}

**public** **void** push(**int** entry) **throws** Exception {

**if** (**this**.isStackFull()) {

**throw** **new** Exception("Stack is already full. Can not add element.");

}

System.***out***.println("Adding: " + entry);

**this**.stack[++top] = entry;

}

**public** **int** pop() **throws** Exception {

**if** (**this**.isStackEmpty()) {

**throw** **new** Exception("Stack is empty. Can not remove element.");

}

**int** entry = **this**.stack[top--];

System.***out***.println("Removed entry: " + entry);

**return** entry;

}

**public** **int** peek() {

**return** stack[top];

}

**public** **boolean** isStackEmpty() {

**return** (top == -1);

}

**public** **boolean** isStackFull() {

**return** (top == size - 1);

}

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

StackEx stack = **new** StackEx(5);

**try** {

stack.push(4);

stack.push(8);

stack.push(3);

stack.push(89);

stack.pop();

stack.push(34);

stack.push(45);

stack.push(78);

} **catch** (Exception e) {

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

}

**try** {

stack.pop();

stack.pop();

stack.pop();

stack.pop();

stack.pop();

stack.pop();

} **catch** (Exception e) {

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

}

}

}

**public** **class** DynamicStackEx {

**private** **int** size;

**private** **int**[] stack;

**private** **int** top;

**public** DynamicStackEx(**int** size) {

**this**.size = size;

**this**.stack = **new** **int**[size];

**this**.top = -1;

}

**public** **void** push(**int** entry) {

**if** (**this**.isStackFull()) {

System.***out***.println(("Stack is full. Increasing the capacity."));

**this**.increaseStackCapacity();

}

System.***out***.println("Adding: " + entry);

**this**.stack[++top] = entry;

}

**public** **int** pop() **throws** Exception {

**if** (**this**.isStackEmpty()) {

**throw** **new** Exception("Stack is empty. Can not remove element.");

}

**int** entry = **this**.stack[top--];

System.***out***.println("Removed entry: " + entry);

**return** entry;

}

**public** **long** peek() {

**return** stack[top];

}

**private** **void** increaseStackCapacity() {

**int**[] newStack = **new** **int**[**this**.size \* 2];

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

newStack[i] = **this**.stack[i];

}

**this**.stack = newStack;

**this**.size = **this**.size \* 2;

}

**public** **boolean** isStackEmpty() {

**return** (top == -1);

}

**public** **boolean** isStackFull() {

**return** (top == size - 1);

}

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

DynamicStackEx stack = **new** DynamicStackEx(2);

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

stack.push(i);

}

**for** (**int** i = 1; i < 4; i++) {

**try** {

stack.pop();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** **class** QueueEx {

**protected** **int** queue[];

**protected** **int** front, rear, size, count;

**public** QueueEx(**int** size) {

**this**.size = size;

count = 0;

queue = **new** **int**[size];

front = 0;

rear = 0;

}

**public** **boolean** isEmpty() {

**return** count == 0;

}

**public** **boolean** isFull() {

**return** count == size;

}

**public** **int** peek() {

**if** (isEmpty())

**throw** **new** NoSuchElementException("Underflow Exception");

**return** queue[front];

}

**public** **void** enqueue(**int** element) {

**if** (isFull())

System.***out***.println("Queue is full");

**if** (rear == size)

rear = 0;

queue[rear++] = element;

System.***out***.println("enqueue = " + element);

count++;

}

**public** **int** dequeue() {

**if** (isEmpty()) {

System.***out***.println("Queue is empty");

**return** -1;

}

**int** element = queue[front++];

**if** (front == size)

front = 0;

count--;

System.***out***.println("dequeue = " + element);

**return** element;

}

**public** **void** display() {

System.***out***.print("\nDisplay Queue = ");

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

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

}

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

}

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

QueueEx queueEx = **new** QueueEx(5);

queueEx.display();

queueEx.enqueue(10);

queueEx.enqueue(20);

queueEx.enqueue(30);

queueEx.display();

queueEx.dequeue();

queueEx.display();

queueEx.enqueue(40);

queueEx.enqueue(50);

queueEx.display();

queueEx.dequeue();

queueEx.dequeue();

queueEx.display();

queueEx.dequeue();

queueEx.dequeue();

queueEx.display();

queueEx.dequeue();

queueEx.display();

}

}

**public** **class** DynamicQueueEx {

**private** **int** capacity = 2;

**int** queue[];

**int** front = 0;

**int** rear = -1;

**int** size = 0;

**public** DynamicQueueEx() {

queue = **new** **int**[**this**.capacity];

}

**public** **void** enqueue(**int** item) {

**if** (isQueueFull()) {

System.***out***.println("Queue is full, increase capacity...");

increaseCapacity();

}

rear++;

**if** (rear >= queue.length && size != queue.length) {

rear = 0;

}

queue[rear] = item;

size++;

System.***out***.println("Adding: " + item);

}

**public** **void** dequeue() {

**if** (isQueueEmpty()) {

System.***out***

.println("Underflow ! Unable to remove element from Queue");

} **else** {

front++;

**if** (front > queue.length - 1) {

System.***out***.println("removed: " + queue[front - 1]);

front = 0;

} **else** {

System.***out***.println("removed: " + queue[front - 1]);

}

size--;

}

}

**public** **boolean** isQueueFull() {

**boolean** status = **false**;

**if** (size == queue.length) {

status = **true**;

}

**return** status;

}

**public** **boolean** isQueueEmpty() {

**boolean** status = **false**;

**if** (size == 0) {

status = **true**;

}

**return** status;

}

**private** **void** increaseCapacity() {

// create new array with double size as the current one.

**int** newCapacity = **this**.queue.length \* 2;

**int**[] newArr = **new** **int**[newCapacity];

// copy elements to new array, copy from rear to front

**int** tmpFront = front;

**int** index = -1;

**while** (**true**) {

newArr[++index] = **this**.queue[tmpFront];

tmpFront++;

**if** (tmpFront == **this**.queue.length) {

tmpFront = 0;

}

**if** (size == index + 1) {

**break**;

}

}

// make new array as queue

**this**.queue = newArr;

System.***out***.println("New array capacity: " + **this**.queue.length);

// reset front & rear values

**this**.front = 0;

**this**.rear = index;

}

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

DynamicQueueEx queue = **new** DynamicQueueEx();

queue.enqueue(4);

queue.dequeue();

queue.enqueue(56);

queue.enqueue(2);

queue.enqueue(67);

queue.dequeue();

queue.enqueue(24);

queue.enqueue(98);

queue.dequeue();

queue.dequeue();

queue.dequeue();

queue.enqueue(435);

queue.dequeue();

queue.dequeue();

}

}

**public** **class** LinkedListEmptyException **extends** RuntimeException {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** LinkedListEmptyException() {

**super**();

}

**public** LinkedListEmptyException(String message) {

**super**(message);

}

}

**public** **class** Node {

**public** **int** data; // data in Node.

**public** Node next; // points to next Node in list.

**public** Node(**int** data) {

**this**.data = data;

}

**public** **void** displayNode() {

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

}

}

**class** LinkedListFirst {

**private** Node first; // ref to first link on list

**public** LinkedListFirst() {

first = **null**;

}

/\*\*

\* Insert New Node at first position in Singly LinkedList

\*/

**public** **void** insertFirst(**int** data) {

Node newNode = **new** Node(data); // Creation of New Node.

newNode.next = first; // newLink ---> old first

first = newNode; // first ---> newNode

}

/\*\*

\* Deletes first Node of Singly LinkedList

\*/

**public** Node deleteFirst() {

**if** (first == **null**) { // means LinkedList in empty, throw exception.

**throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");

}

Node tempNode = first; // save reference to first Node in tempNode- so

// that we could return saved reference.

first = first.next; // delete first Node (make first point to second

// node)

**return** tempNode; // return tempNode (i.e. deleted Node)

}

/\*\*

\* Display Singly LinkedList

\*/

**public** **void** displayLinkedList() {

System.***out***.print("Displaying LinkedList [first--->last]: ");

Node tempDisplay = first; // start at the beginning of linkedList

**while** (tempDisplay != **null**) { // Executes until we don't find end of

// list.

tempDisplay.displayNode();

tempDisplay = tempDisplay.next; // move to next Node

}

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

}

}

**public** **class** SinglyLinkedListInsertDeleteFirstEx {

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

LinkedListFirst linkedList = **new** LinkedListFirst(); // creation of Linked List

linkedList.insertFirst(11);

linkedList.insertFirst(21);

linkedList.insertFirst(59);

linkedList.insertFirst(14);

linkedList.insertFirst(39);

linkedList.displayLinkedList(); // display LinkedList

System.***out***.print("Deleted Nodes: ");

Node deletedNode = linkedList.deleteFirst(); // delete Node

deletedNode.displayNode(); // display deleted Node.

deletedNode = linkedList.deleteFirst(); // delete Node.

deletedNode.displayNode(); // display deleted Node.

System.***out***.println();// sysout used to format output

linkedList.displayLinkedList(); // Again display LinkedList

}

}

/\*

\* OUTPUT

\*

\* Displaying LinkedList [first--->last]: 39 14 59 21 11 Deleted Nodes: 39 14

\* Displaying LinkedList [first--->last]: 59 21 11

\*

\*/

**class** LinkedListLast {

**private** Node first; // ref to first link on list

**public** LinkedListLast() {

first = **null**;

}

/\*\*

\* Inserts new Node at last of Singly LinkedList.

\*/

**public** **void** insertLast(**int** data) {

Node newNode = **new** Node(data); // Creation of New Node.

**if** (first == **null**) { // means LinkedList is empty, make first point to

// new Node.

first = newNode; // first ---> newNode

**return**;

}

Node tempNode = first; // save reference to first Node in tempNode- so

// that we could return saved reference.

**while** (tempNode.next != **null**) { // Executes until we don't find last

// Node of LinkedList.

// If next of some Node is pointing to

// null, that means it's a last Node.

tempNode = tempNode.next; // move to next Node.

}

tempNode.next = newNode; // make last's Node next point to new Node

}

/\*\*

\* Deletes last Node from Singly LinkedList

\*/

**public** Node deleteLast() {

// Case1: when there is no element in LinkedList

**if** (first == **null**) { // means LinkedList in empty, throw exception.

**throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");

}

// Case2: when there is only one element in LinkedList

**if** (first.next == **null**) { // means LinkedList consists of only one

// element, delete that.

Node tempNode = first; // save reference to first Node in tempNode-

// so that we could return saved reference.

first = first.next; // delete firstNode (make first point to

// secondNode)

**return** tempNode; // return deleted Node.

}

// Case3: when there are atLeast two elements in LinkedList

Node previous = **null**;

Node current = first;

**while** (current.next != **null**) {// Executes until we don't find last Node

// of LinkedList.

// If next of some Node is pointing to

// null, that means it's a last Node.

previous = current;

current = current.next; // move to next node.

}

previous.next = **null**; // Now, previous is pointing to second last Node

// of LinkiedList,

// make it point to null [it byPasses current

// Node(last Node of LinkedList) which was in

// between]

**return** current;

}

/\*\*

\* Display LinkedList

\*/

**public** **void** displayLinkedList() {

System.***out***.print("Displaying LinkedList [first--->last]: ");

Node tempDisplay = first; // start at the beginning of linkedList

**while** (tempDisplay != **null**) { // Executes until we don't find end of

// list.

tempDisplay.displayNode();

tempDisplay = tempDisplay.next; // move to next Node

}

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

}

}

**public** **class** SinglyLinkedListInsertDeleteLastEx {

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

LinkedListLast linkedList = **new** LinkedListLast(); // creation of Linked

// List

linkedList.insertLast(11);

linkedList.insertLast(21);

linkedList.insertLast(59);

linkedList.insertLast(14);

linkedList.insertLast(39);

linkedList.displayLinkedList(); // display LinkedList

System.***out***.print("Deleted Nodes: ");

Node deletedNode = linkedList.deleteLast(); // delete Node

deletedNode.displayNode(); // display deleted Node.

deletedNode = linkedList.deleteLast(); // delete Node

deletedNode.displayNode(); // display deleted Node.

System.***out***.println();// sysout used to format output

linkedList.displayLinkedList(); // Again display LinkedList

}

}

/\*

\* OUTPUT

\*

\* Displaying LinkedList [first--->last]: 11 21 59 14 39 Deleted Nodes: 39 14

\* Displaying LinkedList [first--->last]: 11 21 59

\*

\*/

**class** LinkedListBetween {

**private** Node first; // ref to first link on list

/\*\*

\* LinkedList constructor

\*/

**public** LinkedListBetween() {

first = **null**;

}

/\*\*

\* Insert Node in Sorted LinkedList (in between of other Nodes). Note:-

\* Sorted LinkedList is arranged in ascending order.

\*/

**public** **void** insertNodeInSortedLinkedList(**int** data) {

Node newNode = **new** Node(data);

// Case1: when there is no element in LinkedList

**if** (first == **null**) { // means LinkedList is empty, make first point to

// new Node.

first = newNode; // first ---> newNode

System.***out***.println("Node with data=" + newNode.data + " inserted at first.");

**return**;

}

// Case2: when newNode should be placed at first.

**if** (first.data >= newNode.data) {

newNode.next = first;

first = newNode; // first ---> newNode

System.***out***.println("Node with data=" + newNode.data + " inserted at first Node, beacuse newNode is smallest of existing Nodes.");

**return**;

}

// Case3: when newNode should be at some position other than first.

Node current = first;

Node previous = **null**;

**while** (current != **null**) {

**if** (current.data < newNode.data) {

**if** (current.next == **null**) { // means current is last node,

// insert new node after current.

current.next = newNode;

System.***out***.println("Node with data=" + newNode.data + " inserted successfully at last of LinkedList.");

**return**;

}

previous = current;

current = current.next; // move to next node.

} **else** {

newNode.next = previous.next; // make new node's next point to

// previous node's next

previous.next = newNode; // make previous nodes' next point to

// new node.

System.***out***.println("Node with data=" + newNode.data + " inserted successfully in middle of LinkedList.");

**return**;

}

}

}

/\*\*

\* Display Sorted Singly LinkedList

\*/

**public** **void** displayLinkedList() {

System.***out***.print("Displaying LinkedList (first--->last): ");

Node tempDisplay = first; // start at the beginning of linkedList

**while** (tempDisplay != **null**) { // Executes until we don't find end of

// list.

tempDisplay.displayNode();

tempDisplay = tempDisplay.next; // move to next Node

}

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

}

}

**public** **class** SortedSinglyLinkedListInsertNodeBetweenEx {

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

LinkedListBetween linkedList = **new** LinkedListBetween(); // creation of

// Linked List

linkedList.displayLinkedList(); // display LinkedList

linkedList.insertNodeInSortedLinkedList(92);

linkedList.insertNodeInSortedLinkedList(20);

linkedList.insertNodeInSortedLinkedList(19);

linkedList.insertNodeInSortedLinkedList(29);

linkedList.insertNodeInSortedLinkedList(99);

linkedList.displayLinkedList(); // Again display LinkedList

}

}

/\*

\* OUTPUT

\*

\* Displaying LinkedList [first--->last]: Node with data=92 insereted at first.

\* Node with data=20 inserted at first Node, beacuse newNode is smallest of

\* existing Nodes. Node with data=19 inserted at first Node, beacuse newNode is

\* smallest of existing Nodes. Node with data=29 inserted successfully in middle

\* of LinkedList. Node with data=99 inserted successfully at last of LinkedList.

\* Displaying LinkedList [first--->last]: 19 20 29 92 99

\*

\*/

**class** LinkedListDeleteSpecificNode {

**private** Node first; // ref to first link on list

/\*\*

\* LinkedList constructor

\*/

**public** LinkedListDeleteSpecificNode() {

first = **null**;

}

/\*\*

\* Insert New Node at first position

\*/

**public** **void** insertFirst(**int** data) {

Node newNode = **new** Node(data); // Creation of New Node.

newNode.next = first; // newLink ---> old first

first = newNode; // first ---> newNode

}

/\*\*

\* Method deletes specific Node from Singly LinkedList in java.

\*/

**public** Node deleteSpecificNode(**int** deleteKey) {

// Case1: when there is no element in LinkedList

**if** (first == **null**) { // means LinkedList in empty, throw exception.

**throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");

}

// Case2: when there is only one element in LinkedList- check whether we

// have to delete that Node or not.

**if** (first.data == deleteKey) { // means LinkedList consists of only one

// element, delete that.

Node tempNode = first; // save reference to first Node in tempNode-

// so that we could return saved reference.

first = first.next;

System.***out***.println("Node with data=" + tempNode.data + " was found on first and has been deleted.");

**return** tempNode; // return deleted Node.

}

// Case3: when there are atLeast two elements in LinkedList

Node previous = **null**;

Node current = first;

**while** (current != **null**) {

**if** (current.data == deleteKey) {

System.***out***.println("Node with data=" + current.data + " has been deleted.");

previous.next = current.next; // make previous node's next point

// to current node's next.

**return** current; // return deleted Node.

} **else** {

**if** (current.next == **null**) { // Means Node wasn't found.

System.***out***.println("Node with data=" + deleteKey + " wasn't found for deletion.");

**return** **null**;

}

previous = current;

current = current.next; // move to next node.

}

}

**return** **null**;

}

/\*\*

\* Display Singly LinkedList

\*/

**public** **void** displayLinkedList() {

System.***out***.print("Displaying LinkedList [first--->last]: ");

Node tempDisplay = first; // start at the beginning of linkedList

**while** (tempDisplay != **null**) { // Executes until we don't find end of

// list.

tempDisplay.displayNode();

tempDisplay = tempDisplay.next; // move to next Node

}

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

}

}

**public** **class** SinglyLinkedListDeleteNodeEx {

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

LinkedListDeleteSpecificNode linkedList = **new** LinkedListDeleteSpecificNode(); // creation

// of

// Linked

// List

linkedList.insertFirst(92);

linkedList.insertFirst(20);

linkedList.insertFirst(19);

linkedList.insertFirst(29);

linkedList.displayLinkedList(); // display LinkedList

linkedList.deleteSpecificNode(29);

linkedList.deleteSpecificNode(11);

linkedList.displayLinkedList(); // Again display LinkedList

}

}

/\*OUTPUT

Displaying LinkedList [first--->last]: 29 19 20 92

Node with data=92 has been deleted.

Node with data=11 wasn't found for deletion.

Displaying LinkedList [first--->last]: 29 19 20

\*/

**class** LinkedList {

**private** Node first; // ref to first link on list

/\*\*

\* Singly LinkedList constructor

\*/

**public** LinkedList() {

first = **null**;

}

/\*\*

\* REVERSE linkedList.

\*/

**public** **void** reverseLinkedList() {

// Using 3 pointers for reversing LinkedList.

Node previousNode = **null**;

Node currentNode = first;

Node nextNode = first;

**while** (nextNode != **null**) {

nextNode = nextNode.next; // make nextNode point to next node.

currentNode.next = previousNode; // make current node's next point

// to previous node.

previousNode = currentNode; // make previousNode point to

// currentNode.

currentNode = nextNode; // make currentNode point to nextNode.

}

// by this stage of program all the nodes are pointing to previous

// nodes(which has helped us generating reverse of LinkedList.)

first = previousNode; // now make first point to previous node(i.e. last

// node).

System.***out***.println("LinkedList has been reversed successfully.");

}

/\*\*

\* Insert New Node at first position

\*/

**public** **void** insertFirst(**int** data) {

Node newNode = **new** Node(data); // Creation of New Node.

newNode.next = first; // newLink ---> old first

first = newNode; // first ---> newNode

}

/\*\*

\* Display Singly LinkedList

\*/

**public** **void** displayLinkedList() {

System.***out***.print("Displaying LinkedList [first--->last]: ");

Node tempDisplay = first; // start at the beginning of linkedList

**while** (tempDisplay != **null**) { // Executes until we don't find end of

// list.

tempDisplay.displayNode();

tempDisplay = tempDisplay.next; // move to next Node

}

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

}

}

**public** **class** SinglyLinkedListReverseEx {

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

LinkedList linkedList = **new** LinkedList();

linkedList.insertFirst(4);

linkedList.insertFirst(3);

linkedList.insertFirst(2);

linkedList.insertFirst(1);

linkedList.displayLinkedList();

linkedList.reverseLinkedList(); // REVERSE LinkedList

linkedList.displayLinkedList();

}

}

/\*OUTPUT

Displaying LinkedList [first--->last]: 1 2 3 4

LinkedList has been reversed successfully.

Displaying LinkedList [first--->last]: 4 3 2 1

\*/

**public** **class** DisplayInRequiedFormat {

**public** **static** **void** main(String... args) {

**char** matrix[][] = **new** **char**[][] { { 'A', 'A', 'A' }, { 'B', 'B', 'B' },

{ 'C', 'C', 'C' } };

**int** rows = matrix.length;

**int** columns = matrix[0].length;

**char** outputMatrix[][] = **new** **char**[columns][rows];

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++)

outputMatrix[j][i] = matrix[i][j];

}

**for** (**int** i = 0; i < columns; i++) {

**for** (**int** j = 0; j < rows; j++)

System.***out***.print(outputMatrix[i][j] + " ");

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

}

}

}

/\*OUTPUT

A B C

A B C

A B C

\*/

**public** **class** MatrixAddition {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows in matrix : "); // rows and

// columns in

// matrix1 and

// matrix2 must

// be same for

// addition.

**int** rows = scanner.nextInt();

System.***out***.print("Enter number of columns in matrix : ");

**int** columns = scanner.nextInt();

**int**[][] matrix1 = **new** **int**[rows][columns];

**int**[][] matrix2 = **new** **int**[rows][columns];

System.***out***.println("Enter the elements in first matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix1[i][j] = scanner.nextInt();

}

}

System.***out***.println("Enter the elements in second matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix2[i][j] = scanner.nextInt();

}

}

// addition of matrices.

**int**[][] resultMatix = **new** **int**[rows][columns];

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

resultMatix[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

System.***out***.println("\nFirst matrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix1[i][j] + " ");

}

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

}

System.***out***.println("\nSecond matrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix2[i][j] + " ");

}

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

}

System.***out***.println("\nThe sum of the two matrices is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(resultMatix[i][j] + " ");

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter number of rows in matrix : 2

Enter number of columns in matrix : 2

Enter the elements in first matrix :

7

2

5

3

Enter the elements in second matrix :

2

1

3

1

First matrix is :

7 2

5 3

Second matrix is :

2 1

3 1

The sum of the two matrices is :

9 3

8 4

\*/

**public** **class** MatrixSubtraction {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows in matrix : "); // rows and

// columns in

// matrix1 and

// matrix2 must

// be same for

// subtraction.

**int** rows = scanner.nextInt();

System.***out***.print("Enter number of columns in matrix : ");

**int** columns = scanner.nextInt();

**int**[][] matrix1 = **new** **int**[rows][columns];

**int**[][] matrix2 = **new** **int**[rows][columns];

System.***out***.println("Enter the elements in first matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix1[i][j] = scanner.nextInt();

}

}

System.***out***.println("Enter the elements in second matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix2[i][j] = scanner.nextInt();

}

}

// Subtraction of matrices.

**int**[][] resultMatix = **new** **int**[rows][columns];

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

resultMatix[i][j] = matrix1[i][j] - matrix2[i][j];

}

}

System.***out***.println("\nFirst matrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix1[i][j] + " ");

}

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

}

System.***out***.println("\nSecond matrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix2[i][j] + " ");

}

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

}

System.***out***.println("\nThe subtraction of the two matrices is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(resultMatix[i][j] + " ");

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter number of rows in matrix : 2

Enter number of columns in matrix : 2

Enter the elements in first matrix :

7

2

5

3

Enter the elements in second matrix :

2

1

3

1

First matrix is :

7 2

5 3

Second matrix is :

2 1

3 1

The subtraction of the two matrices is :

5 1

2 2

\*/

**public** **class** MatrixMultiplication {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows in first matrix : ");

**int** rowsMatrix1 = scanner.nextInt();

System.***out***

.print("Enter number of columns in first matrix / rows in matrix2: ");

**int** columnsMatrix1RowsMatrix2 = scanner.nextInt(); // variable name used

// for understanding

// convenience,

// because columns

// in matrix1 = rows

// in matrix2

System.***out***.print("Enter number of columns in second matrix : ");

**int** columnsMatrix2 = scanner.nextInt();

**int**[][] matrix1 = **new** **int**[rowsMatrix1][columnsMatrix1RowsMatrix2];

**int**[][] matrix2 = **new** **int**[columnsMatrix1RowsMatrix2][columnsMatrix2];

System.***out***.println("Enter the first matrix in elements :");

**for** (**int** i = 0; i < matrix1.length; i++) {

**for** (**int** j = 0; j < matrix1[0].length; j++) {

matrix1[i][j] = scanner.nextInt();

}

}

System.***out***.println("Enter the second matrix elements:");

**for** (**int** i = 0; i < matrix2.length; i++) {

**for** (**int** j = 0; j < matrix2[0].length; j++) {

matrix2[i][j] = scanner.nextInt();

}

}

// Multiply matrices

**int**[][] productMatrix = **new** **int**[rowsMatrix1][columnsMatrix2];

**for** (**int** i = 0; i < rowsMatrix1; i++) {

**for** (**int** j = 0; j < columnsMatrix2; j++) {

**for** (**int** k = 0; k < columnsMatrix1RowsMatrix2; k++) { // columns

// in

// matrix1=

// rows

// in

// matrix2

productMatrix[i][j] = productMatrix[i][j] + matrix1[i][k]

\* matrix2[k][j];

}

}

}

System.***out***.println("\nFirst matrix is : ");

**for** (**int** i = 0; i < rowsMatrix1; i++) {

**for** (**int** j = 0; j < columnsMatrix1RowsMatrix2; j++) {

System.***out***.print(matrix1[i][j] + " ");

}

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

}

System.***out***.println("\nSecond matrix is : ");

**for** (**int** i = 0; i < columnsMatrix1RowsMatrix2; i++) {

**for** (**int** j = 0; j < columnsMatrix2; j++) {

System.***out***.print(matrix2[i][j] + " ");

}

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

}

System.***out***.println("\nProduct of matrix1 and matrix2 is");

**for** (**int** i = 0; i < rowsMatrix1; i++) {

**for** (**int** j = 0; j < columnsMatrix2; j++) {

System.***out***.print(productMatrix[i][j] + " ");

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter number of rows in first matrix : 2

Enter number of columns in first matrix / rows in matrix2: 3

Enter number of columns in second matrix : 2

Enter the first matrix in elements :

1

2

3

4

5

6

Enter the second matrix elements:

7

8

9

10

11

12

First matrix is :

1 2 3

4 5 6

Second matrix is :

7 8

9 10

11 12

Product of matrix1 and matrix2 is:

58 64

139 154

\*/

**public** **class** SpiralMatrix {

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

System.***out***.println("Enter The Value For N :");

Scanner sc = **new** Scanner(System.***in***);

**int** n = sc.nextInt();

**int**[][] spiral = **new** **int**[n][n];

**int** value = 1;

**int** minCol = 0;

**int** maxCol = n - 1;

**int** minRow = 0;

**int** maxRow = n - 1;

**while** (value <= n \* n) {

**for** (**int** i = minCol; i <= maxCol; i++) {

spiral[minRow][i] = value;

value++;

}

**for** (**int** i = minRow + 1; i <= maxRow; i++) {

spiral[i][maxCol] = value;

value++;

}

**for** (**int** i = maxCol - 1; i >= minCol; i--) {

spiral[maxRow][i] = value;

value++;

}

**for** (**int** i = maxRow - 1; i >= minRow + 1; i--) {

spiral[i][minCol] = value;

value++;

}

minCol++;

minRow++;

maxCol--;

maxRow--;

}

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

**for** (**int** j = 0; j < spiral.length; j++) {

System.***out***.print(spiral[i][j] + "\t");

}

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

}

sc.close();

}

}

**public** **class** SpiralMatrix1 {

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

System.***out***.println("Enter The Value For N :");

// Scanner sc = new Scanner(System.in);

//

// int n = sc.nextInt();

**int** n = 4;

**int**[][] spiral = **new** **int**[n][n];

**int** count = 0;

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

**int** k = n - 1;

**for** (**int** j = 0; j < n; j++) {

**if** (i % 2 != 0) {

spiral[i][k] = ++count;

k--;

} **else** {

spiral[i][j] = ++count;

}

}

}

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

**for** (**int** j = 0; j < n; j++) {

System.***out***.print(spiral[i][j] + "\t");

}

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

}

// sc.close();

}

}

/\*

Enter The Value For N :

1 2 3 4

8 7 6 5

9 10 11 12

16 15 14 13

\*/

**public** **class** SumOfBothDiagonals {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows/columns in matrix : "); // rows

// and

// columns

// in

// matrix

// must

// be

// same.

**int** rows = scanner.nextInt();

**int** columns = rows;

**int**[][] matrix = **new** **int**[rows][rows];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// Logic to calculate sum of diagonal1

**int** sumOfDiagonal1 = 0;

**for** (**int** i = 0, j = 0; i < rows && j < columns; i++, j++) {

sumOfDiagonal1 = sumOfDiagonal1 + matrix[i][j];

}

// Logic to calculate sum of diagonal2

**int** sumOfDiagonal2 = 0;

**for** (**int** i = 0, j = columns - 1; i < rows && j >= 0; i++, j--) {

sumOfDiagonal2 = sumOfDiagonal2 + matrix[i][j];

}

System.***out***.println("\nMatrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("sum of diagonal1 elements=" + sumOfDiagonal1

+ ", sum of diagonal2 elements =" + sumOfDiagonal2);

System.***out***.println("sum of diagonal1 and diagonal2 elements is: "

+ (sumOfDiagonal1 + sumOfDiagonal2));

scanner.close();

}

}

/\*OUTPUT

Enter number of rows/columns in matrix : 4

Enter the elements in matrix :

1

2

3

4

5

6

7

8

8

7

6

5

4

3

2

1

Matrix is :

1 2 3 4

5 6 7 8

8 7 6 5

4 3 2 1

sum of diagonal1 elements=14, sum of diagonal2 elements =22

sum of diagonal1 and diagonal2 elements is: 36

\*/

**public** **class** SumOfElementsAboveDiagonal {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows/columns in matrix : "); // rows

// and

// columns

// in

// matrix

// must

// be

// same.

**int** rows = scanner.nextInt();

**int** columns = rows;

**int**[][] matrix = **new** **int**[rows][rows];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// Logic to calculate sum of elements above diagonal.

**int** sum = 0;

**for** (**int** j = 1; j < columns; j++) {

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

sum = sum + matrix[i][j];

}

}

System.***out***.println("\nMatrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("sum of elements above diagonal is: " + sum);

scanner.close();

}

}

/\*OUTPUT

Enter number of rows/columns in matrix : 4

Enter the elements in matrix :

1

2

3

4

5

6

7

8

8

7

6

5

4

3

2

1

Matrix is :

1 2 3 4

5 6 7 8

8 7 6 5

4 3 2 1

sum of elements above diagonal is: 29

\*/

**public** **class** SumOfElementsBelowDiagonal {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows/columns in matrix : "); // rows

// and

// columns

// in

// matrix

// must

// be

// same.

**int** rows = scanner.nextInt();

**int** columns = rows;

**int**[][] matrix = **new** **int**[rows][rows];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// Logic to calculate sum of elements below diagonal.

**int** sum = 0;

**for** (**int** i = 1; i < rows; i++) {

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

sum = sum + matrix[i][j];

}

}

System.***out***.println("\nMatrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("sum of elements below diagonal is: " + sum);

scanner.close();

}

}

/\*OUTPUT

Enter number of rows/columns in matrix : 4

Enter the elements in matrix :

1

2

3

4

5

6

7

8

8

7

6

5

4

3

2

1

Matrix is :

1 2 3 4

5 6 7 8

8 7 6 5

4 3 2 1

sum of elements below diagonal is: 29

\*/

**public** **class** SumOfLowerTriangle {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows/columns in matrix : "); // rows

// and

// columns

// in

// matrix

// must

// be

// same.

**int** rows = scanner.nextInt();

**int** columns = rows;

**int**[][] matrix = **new** **int**[rows][rows];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// Logic to calculate sum of lower triangle.

**int** sum = 0;

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = i; j >= 0; j--) {

sum = sum + matrix[i][j];

}

}

System.***out***.println("\nMatrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("sum of lower triangle is: " + sum);

scanner.close();

}

}

/\*OUTPUT

Enter number of rows/columns in matrix : 4

Enter the elements in matrix :

1

2

3

4

5

6

7

8

8

7

6

5

4

3

2

1

Matrix is :

1 2 3 4

5 6 7 8

8 7 6 5

4 3 2 1

sum of lower triangle is: 43

\*/

**public** **class** SumOfUpperTriangle {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter number of rows/columns in matrix : "); // rows

// and

// columns

// in

// matrix

// must

// be

// same.

**int** rows = scanner.nextInt();

**int** columns = rows;

**int**[][] matrix = **new** **int**[rows][rows];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// Logic to calculate sum of upper triangle.

**int** sum = 0;

**for** (**int** j = 0; j < columns; j++) {

**for** (**int** i = j; i >= 0; i--) {

sum = sum + matrix[i][j];

}

}

System.***out***.println("\nMatrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("sum of upper triangle is: " + sum);

scanner.close();

}

}

/\*OUTPUT

Enter number of rows/columns in matrix : 4

Enter the elements in matrix :

1

2

3

4

5

6

7

8

8

7

6

5

4

3

2

1

Matrix is :

1 2 3 4

5 6 7 8

8 7 6 5

4 3 2 1

sum of upper triangle is: 43

\*/

**public** **class** TransposeMatrix {

**public** **static** **void** main(String... args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("Enter number of rows in matrix : ");

**int** rows = scanner.nextInt();

System.***out***.print("Enter number of columns in matrix : ");

**int** columns = scanner.nextInt();

**int** matrix[][] = **new** **int**[rows][columns];

System.***out***.println("Enter the elements in matrix :");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

matrix[i][j] = scanner.nextInt();

}

}

// transpose matrix

**int** transpose[][] = **new** **int**[columns][rows];

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++)

transpose[j][i] = matrix[i][j];

}

System.***out***.println("\nEntered Matrix is : ");

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < columns; j++) {

System.***out***.print(matrix[i][j] + " ");

}

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

}

System.***out***.println("\nTranspose of entered matrix is : ");

**for** (**int** i = 0; i < columns; i++) {

**for** (**int** j = 0; j < rows; j++)

System.***out***.print(transpose[i][j] + " ");

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

}

scanner.close();

}

}

/\*OUTPUT

Enter number of rows in matrix :

3

Enter number of columns in matrix : 2

Enter the elements in matrix :

1

2

3

4

5

6

Entered Matrix is :

1 2

3 4

5 6

Transpose of entered matrix is :

1 3 5

2 4 6

\*/

**public** **class** Pyramid1 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = 1; j <= i; j++) {

System.***out***.print(j);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

1

12

123

1234

12345

\*/

**public** **class** Pyramid2 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = 1; j <= i; j++) {

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

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

1

22

333

4444

55555

\*/

**public** **class** Pyramid3 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = 1; j <= i; j++) {

System.***out***.print(j);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

12345

1234

123

12

1

\*/

**public** **class** Pyramid4 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = n, k = 1; k <= i; j++, k++) {

System.***out***.print(j);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

56789

5678

567

56

5

\*/

**public** **class** Pyramid5 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = n, k = 1; k <= i; j--, k++) {

System.***out***.print(j);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

54321

5432

543

54

5

\*/

**public** **class** Pyramid6 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = i, k = 1; k <= i; j--, k++) {

System.***out***.print(j);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

54321

4321

321

21

1

\*/

**public** **class** Pyramid7Floyds {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter rows in Floyd's triangle : ");

**int** rows = scanner.nextInt();

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

**int** nextNumber = 1;

**for** (**int** i = 1; i <= rows; i++) {

**for** (**int** j = 1; j <= i; j++) {

System.***out***.print(nextNumber < 10 ? (" " + nextNumber++) : (" " + nextNumber++));

// 2spaces in single digit & 1 space in double digit.

// System.out.format("%3d",nextNumber++ ); //You may use this

// line for formatting as a replacement of above line. (comment

// above line before using this)

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter rows in Floyd's triangle : 5

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

\*/

**public** **class** Pyramid8Pascal {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter no of rows in Pascal's triangle : ");

**int** rows = scanner.nextInt();

**int** nextNumber;

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

**for** (**int** i = 0; i < rows; i++) {

nextNumber = 1;

**for** (**int** k = 0; k < (rows - i) \* 2; k++)

// create (rows-i)\*2 spaces, for initial spacing.

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

**for** (**int** j = 0; j <= i; j++) {

System.***out***.format("%4d", nextNumber); // %4d creates 4 space between number.

nextNumber = nextNumber \* (i - j) / (j + 1);

}

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

}

scanner.close();

}

}

/\* OUTPUT

Enter no of rows in Pascal's triangle : 6

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

\*/

**public** **class** Pyramid9Double {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

**for** (**int** i = 1; i <= n; i++) { // it will create upper half

**for** (**int** j = 1; j <= i; j++) {

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

}

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

}

**for** (**int** i = n; i > 0; i--) { // it will create lower half

**for** (**int** j = 1; j <= i; j++) {

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

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 4

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

\*/

**public** **class** Pyramid10 {

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

**int** j;

**int** rows = 8;

**for** (**int** i = 1; i <= rows; i++) {

**for** (j = 1; j <= rows - i; j++)

// for initial spacing.

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

**for** (**int** k = j + 1; k <= rows; k++)

// creates left half.

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

**for** (**int** k = rows; k > j - 1; k--)

// creates right half.

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

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

}

}

}

/\*OUTPUT

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

**public** **class** Pyramid11 {

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

**int** rows = 8;

**for** (**int** i = 1; i <= rows; i++) {

**for** (**int** j = 1; j <= (rows - i) \* 2; j++)

// create initial spacing.

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

**for** (**int** k = i; k >= 1; k--)

// creates left half.

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

**for** (**int** k = 2; k <= i; k++)

// creates right half.

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

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

}

}

}

/\*OUTPUT

1

2 1 2

3 2 1 2 3

4 3 2 1 2 3 4

5 4 3 2 1 2 3 4 5

6 5 4 3 2 1 2 3 4 5 6

7 6 5 4 3 2 1 2 3 4 5 6 7

8 7 6 5 4 3 2 1 2 3 4 5 6 7 8

\*/

**public** **class** Pyramid12 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**int** k = i;

**for** (**int** j = 1; j <= n; j++) {

System.***out***.print((j <= n - i ? " " : k--));

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

1

21

321

4321

54321

\*/

**public** **class** Pyramid13 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**int** k = i;

**for** (**int** j = 1; j <= n; j++) {

**if** (j < i)

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

**else**

System.***out***.print(k++);

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

12345

2345

345

45

5

\*/

**public** **class** Pyramid14 {

**public** **static** **void** main(String... args) {

**int** ar[][] = **new** **int**[5][]; // jagged array

**int** start = 0;

**int** prev = 0;

**int** sum = 1; // keep initial sum as 0

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

ar[i] = **new** **int**[i + 1];

prev = 0;

**for** (**int** j = 0; j < ar[i].length; j++) {

**if** (i > 0 && j > 0) {

prev = ar[i][j - 1];

start = ar[i - 1][j - 1];

sum = start + prev;

ar[i][j] = sum;

} **else** {

ar[i][j] = sum;

}

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

}

start = sum; // assign sum to start

System.***out***.println(); // for new line

} // end for loop

}

}

/\*

1

1 2

2 3 5

5 7 10 15

15 20 27 37 52

\*/

**public** **class** Pyramid15 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter n : ");

**int** n = scanner.nextInt();

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

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

**for** (**int** j = 1, k = i; j <= i; j++) {

System.***out***.print(k \* j + " ");

}

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

}

scanner.close();

}

}

/\*OUTPUT

Enter n : 5

1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

\*/

**public** **class** PreOrderTraversalOfBinaryTree {

/\*\*

\* Create Node of Binary Tree

\*/

**public** **static** **class** BinaryTreeNode {

// Left and right node of Binary tree

BinaryTreeNode leftNode;

BinaryTreeNode rightNode;

// Data of BinaryTree

**int** data;

BinaryTreeNode(**int** data) {

**this**.data = data;

}

}

/\*\*

\* This method will do pre-order Traversal of binary tree in java. Method

\* will pass leftNode of binary tree recursively Once leftNode child is

\* done, pass rightNode of binary tree recursively

\*/

**public** **void** preorderTraversalMethod(BinaryTreeNode binaryTreeNode) {

// Continue if binaryTreeNode is not null

**if** (binaryTreeNode != **null**) {

// Display data of current binaryTreeNode.

System.***out***.print(binaryTreeNode.data + " ");

// pass leftNode of binary tree recursively

preorderTraversalMethod(binaryTreeNode.leftNode);

// pass rightNode of binary tree recursively

preorderTraversalMethod(binaryTreeNode.rightNode);

}

}

/\*\*

\* Main method to perform PreOrder Traversal of Binary Tree in java

\*/

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

PreOrderTraversalOfBinaryTree preOrderTraversalOfBinaryTree = **new** PreOrderTraversalOfBinaryTree();

// Now, let's start creating nodes of Binary tree

// create node1 (which is ROOT node)

BinaryTreeNode binaryTreeNode1 = **new** BinaryTreeNode(1);

// Left node of node1

BinaryTreeNode node2 = **new** BinaryTreeNode(2);

binaryTreeNode1.leftNode = node2;

// Right node of node1

BinaryTreeNode node3 = **new** BinaryTreeNode(3);

binaryTreeNode1.rightNode = node3;

// Left node of node2

BinaryTreeNode node4 = **new** BinaryTreeNode(4);

node2.leftNode = node4;

// Right node of node2

BinaryTreeNode node5 = **new** BinaryTreeNode(5);

node2.rightNode = node5;

// Left node of node3

BinaryTreeNode node6 = **new** BinaryTreeNode(6);

node3.leftNode = node6;

// Right node of node3

BinaryTreeNode node7 = **new** BinaryTreeNode(7);

node3.rightNode = node7;

System.***out***.println("Pre order Traversal of Binary Tree in java : ");

// Pass root node to pre order traversal method

preOrderTraversalOfBinaryTree.preorderTraversalMethod(binaryTreeNode1);

}

}

/\*

\* Output

\*

\* Pre order Traversal of Binary Tree in java : 1 2 4 5 3 6 7

\*

\*/

**public** **class** InOrderTraversalOfBinaryTree {

/\*\*

\* Create Node of Binary Tree

\*/

**public** **static** **class** BinaryTreeNode {

// Left and right node of Binary tree

BinaryTreeNode leftNode;

BinaryTreeNode rightNode;

// Data of BinaryTree

**int** data;

BinaryTreeNode(**int** data) {

**this**.data = data;

}

}

/\*\*

\* This method will do in-order Traversal of binary tree in java. Method

\* will pass leftNode of binary tree recursively Once leftNode child is

\* done, pass rightNode of binary tree recursively

\*/

**public** **void** inorderTraversalMethod(BinaryTreeNode binaryTreeNode) {

// Continue if binaryTreeNode is not null

**if** (binaryTreeNode != **null**) {

// pass leftNode of binary tree recursively

inorderTraversalMethod(binaryTreeNode.leftNode);

// Display data of current binaryTreeNode.

System.***out***.print(binaryTreeNode.data + " ");

// pass rightNode of binary tree recursively

inorderTraversalMethod(binaryTreeNode.rightNode);

}

}

/\*\*

\* Main method to perform InOrder Traversal of Binary Tree in java

\*/

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

InOrderTraversalOfBinaryTree inOrderTraversalOfBinaryTree = **new** InOrderTraversalOfBinaryTree();

// Now, let's start creating nodes of Binary tree

// create node1 (which is ROOT node)

BinaryTreeNode binaryTreeNode1 = **new** BinaryTreeNode(1);

// Left node of node1

BinaryTreeNode node2 = **new** BinaryTreeNode(2);

binaryTreeNode1.leftNode = node2;

// Right node of node1

BinaryTreeNode node3 = **new** BinaryTreeNode(3);

binaryTreeNode1.rightNode = node3;

// Left node of node2

BinaryTreeNode node4 = **new** BinaryTreeNode(4);

node2.leftNode = node4;

// Right node of node2

BinaryTreeNode node5 = **new** BinaryTreeNode(5);

node2.rightNode = node5;

// Left node of node3

BinaryTreeNode node6 = **new** BinaryTreeNode(6);

node3.leftNode = node6;

// Right node of node3

BinaryTreeNode node7 = **new** BinaryTreeNode(7);

node3.rightNode = node7;

System.***out***.println("In order Traversal of Binary Tree in java : ");

// Pass root node to in order traversal method

inOrderTraversalOfBinaryTree.inorderTraversalMethod(binaryTreeNode1);

}

}

/\*

\* Output

\*

\* In order Traversal of Binary Tree in java : 4 2 5 1 6 3 7

\*

\*/

**public** **class** PostOrderTraversalOfBinaryTree {

/\*\*

\* Create Node of Binary Tree

\*/

**public** **static** **class** BinaryTreeNode {

// Left and right node of Binary tree

BinaryTreeNode leftNode;

BinaryTreeNode rightNode;

// Data of BinaryTree

**int** data;

BinaryTreeNode(**int** data) {

**this**.data = data;

}

}

/\*\*

\* This method will do Post-order Traversal of binary tree in java. Method

\* will pass leftNode of binary tree recursively Once leftNode child is

\* done, pass rightNode of binary tree recursively

\*/

**public** **void** postorderTraversalMethod(BinaryTreeNode binaryTreeNode) {

// Continue if binaryTreeNode is not null

**if** (binaryTreeNode != **null**) { // Visit the node-Printing the node data

// pass leftNode of binary tree recursively

postorderTraversalMethod(binaryTreeNode.leftNode);

// pass rightNode of binary tree recursively

postorderTraversalMethod(binaryTreeNode.rightNode);

// Display data of current binaryTreeNode.

System.***out***.print(binaryTreeNode.data + " ");

}

}

/\*\*

\* Main method to perform PostOrder Traversal of Binary Tree in java

\*/

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

PostOrderTraversalOfBinaryTree postOrderTraversalOfBinaryTree = **new** PostOrderTraversalOfBinaryTree();

// Now, let's start creating nodes of Binary tree

// create node1 (which is ROOT node)

BinaryTreeNode binaryTreeNode1 = **new** BinaryTreeNode(1);

// Left node of node1

BinaryTreeNode node2 = **new** BinaryTreeNode(2);

binaryTreeNode1.leftNode = node2;

// Right node of node1

BinaryTreeNode node3 = **new** BinaryTreeNode(3);

binaryTreeNode1.rightNode = node3;

// Left node of node2

BinaryTreeNode node4 = **new** BinaryTreeNode(4);

node2.leftNode = node4;

// Right node of node2

BinaryTreeNode node5 = **new** BinaryTreeNode(5);

node2.rightNode = node5;

// Left node of node3

BinaryTreeNode node6 = **new** BinaryTreeNode(6);

node3.leftNode = node6;

// Right node of node3

BinaryTreeNode node7 = **new** BinaryTreeNode(7);

node3.rightNode = node7;

System.***out***.println("Post order Traversal of Binary Tree in java : ");

// Pass root node to Post order traversal method

postOrderTraversalOfBinaryTree.postorderTraversalMethod(binaryTreeNode1);

}

}

/\*

\* Output

\*

\* Post order Traversal of Binary Tree in java : 4 5 2 6 7 3 1

\*

\*/

**public** **class** CharacterCountInStringUsingHashMapEx {

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

String inputString = "This is it";

System.***out***.println("inputString is : " + inputString);

System.***out***.print("Output is : ");

*characterCount*(inputString);

}

/\*\*

\* Method calculates count of all characters in inputString.

\*/

**public** **static** **void** characterCount(String inputString) {

Map<Character, Integer> map = **new** LinkedHashMap<Character, Integer>();

**char**[] inputAr = inputString.toCharArray();

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

**char** ch = inputAr[i];

**if** (map.containsKey(ch))

map.put(ch, map.get(ch) + 1);

**else**

map.put(ch, 1);

}

/\*

\* Till this point of program, we have stored all unique characters in

\* map as key & corresponding value representing count of character.

\*/

Iterator<Character> charIterator = map.keySet().iterator();

**while** (charIterator.hasNext()) {

**char** ch = charIterator.next();

System.***out***.print(ch + "=" + map.get(ch) + " ");

}

}

}

/\*

\* OUTPUT

\*

\* inputString is : This is it Output is :T=1 h=1 i=3 s=2 =2 t=1

\*

\*/

**public** **class** CheckContinuityOfString {

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

String inputString = "wbaodralrd";

String findString = "world";

System.***out***.println(*checkContinuity*(inputString, findString) == **true**

? "TRUE" : "FALSE");

}

/\*\*

\* returns true if required condition matches.

\*/

**static** **boolean** checkContinuity(String inputString, String findString) {

**char** inputCh[] = inputString.toCharArray();

**char** findCh[] = findString.toCharArray();

**int** pos = 0;

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

**if** (inputCh[i] == findCh[pos]) {

pos++;

**if** (pos == findCh.length)

**return** **true**;

}

}

**return** **false**;

}

}

/\*

\* OUTPUT

\*

\* TRUE

\*

\*/

**public** **class** ConcatenateTwoArrays {

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

**int**[] array1 = { 1, 2, 3 };

**int**[] array2 = { 4, 5, 6 };

**int** aLen = array1.length;

**int** bLen = array2.length;

**int**[] result = **new** **int**[aLen + bLen];

System.*arraycopy*(array1, 0, result, 0, aLen);

System.*arraycopy*(array2, 0, result, aLen, bLen);

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

}

}

/\*

\* OUTPUT

\*

\* Display concatenated array 1 2 3 4 5 6

\*

\*/

**public** **class** FindFirstRepeatedAndNonRepeatedCharacterInAString {

**static** **void** firstRepeatedNonRepeatedChar(String inputString) {

// Creating a HashMap containing char as a key and occurrences as a

// value

HashMap<Character, Integer> charCountMap = **new** HashMap<Character, Integer>();

// Converting inputString to char array

**char**[] strArray = inputString.toCharArray();

// Checking each char of strArray

**for** (**char** c : strArray) {

**if** (charCountMap.containsKey(c)) {

// If char is present in charCountMap, incrementing it's count

// by 1

charCountMap.put(c, charCountMap.get(c) + 1);

} **else** {

// If char is not present in charCountMap,

// adding this char in charCountMap with 1 as it's value

charCountMap.put(c, 1);

}

}

// checking for first non-repeated character

**for** (**char** c : strArray) {

**if** (charCountMap.get(c) == 1) {

System.***out***.println("First Non-Repeated Character In '" + inputString + "' is '" + c + "'");

**break**;

}

}

// checking for first repeated character

**for** (**char** c : strArray) {

**if** (charCountMap.get(c) > 1) {

System.***out***.println("First Repeated Character In '" + inputString + "' is '" + c + "'");

**break**;

}

}

}

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter the string :");

String input = sc.next();

*firstRepeatedNonRepeatedChar*(input);

sc.close();

}

}

/\*

Enter the string :

JavaConceptOfTheDay

First Non-Repeated Character In 'JavaConceptOfTheDay' is 'J'

First Repeated Character In 'JavaConceptOfTheDay' is 'a'

\*/

**public** **class** FindIndexSumOfElementsWhoseLeftIsEqualToRightEx {

**public** **static** **void** main(String... args) {

**int**[] ar = { 2, 4, 4, 2, 6, 1, 1 };

System.***out***.print("In ar=[");

**for** (**int** n : ar) {

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

}

System.***out***.print("] : ");

**int** findIndex = *findIndex*(ar);

System.***out***

.println(findIndex != -1

? "sum of elements on left(i.e. from index 0 to " + findIndex + ") = sum of elements on right(i.e. from index "

+ (findIndex + 1) + " to " + (ar.length - 1) + ")"

: "No index found where sum of elements on left = sum of elements on right");

}

/\*\*

\* Finds index such that > sum of elements on whose left is equal to sum of

\* elements on right

\*/

**public** **static** **int** findIndex(**int**[] ar) {

**int** leftIndex = 0, rightIndex = ar.length - 1;

**int** leftSum = 0, rightSum = 0;

**while** (leftIndex <= rightIndex) {

**if** (leftSum > rightSum)

rightSum = rightSum + ar[rightIndex--];

**else**

leftSum = leftSum + ar[leftIndex++];

}

**if** (leftSum == rightSum)

**return** rightIndex;

**else**

**return** -1; // appropriate index not found

}

}

/\*

\* OUTPUT

\*

\* In ar=[2 4 4 2 6 1 1 ] : sum of elements on left(i.e. from index 0 to 2) =

\* sum of elements on right(i.e. from index 3 to 6)

\*

\*/

**public** **class** FindSmallestAndLargestNumberInArrayEx {

**static** **int** *ar*[] = { 3, 9, 4, 9, 1, 2, 5, 8, 12, 7 }; // given array

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

**int** smallestNumber = *ar*[0];

**int** largestNumber = *ar*[0];

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

**if** (*ar*[i] > largestNumber)

largestNumber = *ar*[i];

**else** **if** (*ar*[i] < smallestNumber)

smallestNumber = *ar*[i];

}

System.***out***.println("Smallest Number in array is : " + smallestNumber);

System.***out***.println("Largest Number in array is : " + largestNumber);

}

}

/\*

\*

\* Smallest Number in array is : 1

\* Largest Number in array is : 12

\*

\*/

**public** **class** FindSubString {

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

String inputString;

String searchSubString;

**int** foundIndex; // index at which subString is found by our method.

**int** javaApiFoundIndex; // index at which subString is found by JAVA API's indexOf() method.

inputString = "abctigeraCd";

searchSubString = "tiger";

System.***out***.println("TEST1");

foundIndex = *isSubString*(inputString, searchSubString);

System.***out***.println(foundIndex > -1 ? inputString + " contains "

+ searchSubString + " at index " + foundIndex : inputString

+ "doesn't contains " + searchSubString);

javaApiFoundIndex = inputString.indexOf("tiger");

System.***out***.print("Java API's indexOf() method says> ");

System.***out***.println(javaApiFoundIndex > -1 ? inputString + " contains "

+ searchSubString + " at index " + javaApiFoundIndex

: inputString + "doesn't contains " + searchSubString);

inputString = "tigeraCd";

searchSubString = "tiger";

System.***out***.println("\nTEST2");

foundIndex = *isSubString*(inputString, searchSubString);

System.***out***.println(foundIndex > -1 ? inputString + " contains "

+ searchSubString + " at index " + foundIndex : inputString

+ "doesn't contains " + searchSubString);

javaApiFoundIndex = inputString.indexOf("tiger");

System.***out***.print("Java API's indexOf() method says> ");

System.***out***.println(javaApiFoundIndex > -1 ? inputString + " contains "

+ searchSubString + " at index " + javaApiFoundIndex

: inputString + "doesn't contains " + searchSubString);

}

/\*\*

\* Below method returns subString index in given inputString. Returns -1 if

\* subString is not found.

\*/

**public** **static** **int** isSubString(String inputString, String searchSubString) {

**char** inputStringAr[] = inputString.toCharArray();

**char** searchSubStringAr[] = searchSubString.toCharArray();

**int** count = 0; // if count matches searchSubStringAr length,than string is found in string

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

**if** (String.*valueOf*(inputStringAr[i]).equals(

String.*valueOf*(searchSubStringAr[count]))) {

count++; // index of searchSubStringAr

**if** (count == searchSubStringAr.length) {

**return** i - count + 1; // substring found, return starting index of

}

} **else** {

/\*

\* Below if condition comes into role, when some part of

\* searchSubString is present before whole searchSubString. Eg=

\* searchSubString=tiger & inputString="tigtiger" [here, (tig)

\* is present before (tiger)]

\*/

**if** (count > 0) {

count = 0;

i--;

}

}

}

**return** -1; // substring not found.

}

}

/\*OUTPUT

TEST1

abctigeraCd contains tiger at index 3

Java API's indexOf() method says> abctigeraCd contains tiger at index 3

TEST2

tigeraCd contains tiger at index 0

Java API's indexOf() method says> tigeraCd contains tiger at index 0

\*/

**public** **class** FindSumOfAllEvenDigitInString {

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

String s = "java grey243";

// String s = "java64 is12"; //Test string

**char** ch[] = s.toCharArray();

**int** sum = 0;

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

**try** {

**int** x = Integer.*valueOf*(String.*valueOf*(ch[i]));

**if** (x % 2 == 0) {

sum += x;

}

} **catch** (Exception e) {

}

}

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

}

}

/\*

\* OUTPUT

\* 6

\*/

**public** **class** FindTwoMaximumNumbersInArrayEx {

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

**int** ar[] = { 66, 7, 9, 70, 89, 11, 2, 93 };

**int** maximum1 = 0;

**int** Maximum2 = 0;

**for** (**int** i = 0; i < ar.length; i++) { // only one iteration

**if** (maximum1 < ar[i]) {

Maximum2 = maximum1;

maximum1 = ar[i];

} **else** **if** (Maximum2 < ar[i]) {

Maximum2 = ar[i];

}

}

System.***out***.println("Maximum1 : " + maximum1);

System.***out***.println("Maximum2 : " + Maximum2);

}

}

**public** **class** FirstNonRepeatedCharacterInStringEx {

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

String inputString = "this is it";

System.***out***

.println("The first non repeated character in inputString("

+ inputString + ") is : "

+ *firstNonRepeatedCharacter*(inputString));

}

/\*\*

\* Method returns first non-repeating character in inputString. Returns null

\* if there is no non-repeating character in inputString

\*/

**public** **static** Character firstNonRepeatedCharacter(String inputString) {

Map<Character, Integer> map = **new** LinkedHashMap<Character, Integer>();

**char** ar[] = inputString.toCharArray();

**char** ch;

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

ch = ar[i];

**if** (map.containsKey(ch)) // if map already contains this character

// as key, get value corresponding to

// key and increment it.

map.put(ch, map.get(ch) + 1);

**else** // put character in map with value as 1 (showing first

// occurrence of key in string)

map.put(ch, 1);

}

/\*

\* Till this point of program, we have stored all unique characters in

\* map as key & corresponding value representing count of character.

\*/

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

ch = ar[i];

**if** (map.get(ch) == 1) // we have found our first non-repeating

// character in string.

**return** ch;

}

**return** **null**;

}

}

/\*

\* OUTPUT

\*

\* The first non repeated character in inputString(this is it) is : h

\*

\*/

**public** **class** IntersectionOfTwoArrays {

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

String[] inputArray1 = { "ONE", "TWO", "THREE", "FOUR", "FIVE", "FOUR" };

String[] inputArray2 = { "THREE", "FOUR", "FIVE", "SIX", "SEVEN",

"FOUR" };

HashSet<String> set = **new** HashSet<String>();

**for** (**int** i = 0; i < inputArray1.length; i++) {

**for** (**int** j = 0; j < inputArray2.length; j++) {

**if** (inputArray1[i].equals(inputArray2[j])) {

set.add(inputArray1[i]);

}

}

}

System.***out***.println("First Array : " + Arrays.*toString*(inputArray1));

System.***out***.println("Second Array : " + Arrays.*toString*(inputArray2));

System.***out***.println("Common Elements : " + set);

}

}

**public** **class** MergeTwoSortedArraysEx {

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

**int**[] ar1 = { 1, 3, 7, 11, 25 };

**int**[] ar2 = { 2, 5, 8, 22 };

System.***out***.print("Display ar1 : ");

**for** (**int** i = 0; i < ar1.length; i++)

System.***out***.print(ar1[i] + " ");

System.***out***.print("\nDisplay ar2 : ");

**for** (**int** i = 0; i < ar2.length; i++)

System.***out***.print(ar2[i] + " ");

**int** mergedArray[] = *merging*(ar1, ar2); // merging both arrays.

System.***out***.print("\nDisplay merged array: ");

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

System.***out***.print(mergedArray[i] + " ");

}

/\*

\* Method merges two sorted arrays in java.

\*/

**public** **static** **int**[] merging(**int**[] ar1, **int** ar2[]) {

**int** mergedArray[] = **new** **int**[ar1.length + ar2.length];

**int** ar1Index = 0, ar2Index = 0, mergedArrayIndex = 0;

**while** (ar1Index < ar1.length && ar2Index < ar2.length)

**if** (ar1[ar1Index] < ar2[ar2Index])

mergedArray[mergedArrayIndex++] = ar1[ar1Index++];

**else**

mergedArray[mergedArrayIndex++] = ar2[ar2Index++];

**while** (ar1Index < ar1.length)

mergedArray[mergedArrayIndex++] = ar1[ar1Index++];

**while** (ar2Index < ar2.length)

mergedArray[mergedArrayIndex++] = ar2[ar2Index++];

**return** mergedArray;

}

}

/\*

\* OUTPUT

\*

\* Display ar1 : 1 3 7 11

\* Display ar2 : 2 5 8 22

\* Display merged array: 1 2 3 5 7 8 11 22

\*

\*/

**public** **class** NumberInWords {

String one[] = { " ",

" one", " two", " three", " four", " five", " six", " seven", " eight", " Nine", " ten",

" eleven", " twelve", " thirteen", " fourteen", " fifteen", " sixteen", " seventeen", " eighteen", " nineteen" };

String ten[] = { " ", " ",

" twenty", " thirty", " forty", " fifty", " sixty", " seventy", " eighty", " ninety" };

**public** **void** pw(**int** n, String str) {

**if** (n > 19) {

System.***out***.print(ten[n / 10] + " " + one[n % 10]);

} **else** {

System.***out***.print(one[n]);

}

**if** (n > 0)

System.***out***.print(str);

}

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

**int** n = 0;

Scanner scanf = **new** Scanner(System.***in***);

System.***out***.println("Enter an integer number: ");

n = scanf.nextInt();

**if** (n <= 0)

System.***out***.println("Enter numbers greater than 0");

**else** {

NumberInWords inWords = **new** NumberInWords();

inWords.pw((n / 1000000000), " Hundred"); // 100,00,00,000

inWords.pw((n / 10000000) % 100, " crore"); // 1,00,00,000

inWords.pw(((n / 100000) % 100), " lakh");// 1,00,000

inWords.pw(((n / 1000) % 100), " thousand");// 1,000

inWords.pw(((n / 100) % 10), " hundred");// 100

inWords.pw((n % 100), " ");

}

scanf.close();

}

}

**public** **class** NumberIsOddOrEvenEx {

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

**int** number = 53;

**if** ((number & 1) == 0)

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

**else**

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

}

}

/\*

\* OUTPUT

\*

\* 53 is ODD

\*

\*/

**public** **class** OccurrenceOfNumberInCubeEx {

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

**long** num = 1551;

**int** occurrenceOf = 1; // calculate OccurrenceOf this digit in number.

System.***out***.println("Cube of entered number(" + num + ") is: " + num \* num \* num);

System.***out***.println("Occurrence of " + occurrenceOf + " in cube of " + num + " is: " + *calculateOccurrenceOf*(num, occurrenceOf));

}

/\*\*

\* method return OccurrenceOf digit in cube of given number.

\*/

**public** **static** **int** calculateOccurrenceOf(**long** num, **int** occurrenceOf) {

**long** cubeOfNumber = num \* num \* num;

**int** count = 0;

**while** (cubeOfNumber > 0) {

**if** (cubeOfNumber % 10 == occurrenceOf) {

count++;

}

cubeOfNumber = cubeOfNumber / 10;

}

**return** count;

}

}

/\*

\* OUTPUT

\* Cube of entered number(1551) is: 3731087151 Occurrence of 1 in cube of 1551 is: 3

\*/

**public** **class** PemutationOfStringRecursion {

**public** **static** **void** main(String... args) {

String inputString = "XYZ";

System.***out***.println("Permutations of inputString(" + inputString

+ ") are: " + *findPermutation*(inputString));

}

/\*\*

\* method returns permutations of string.

\*/

**public** **static** Set<String> findPermutation(String inputString) {

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

Set<String> set2;

String stringWithoutFirstChar;

**char** firstChar;

**if** (inputString.length() == 0) {

set.add("");

**return** set;

}

firstChar = inputString.charAt(0);

stringWithoutFirstChar = inputString.substring(1);

set2 = *findPermutation*(stringWithoutFirstChar);

**for** (String s : set2) {

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

set.add(*insertCharacter*(s, firstChar, k));

}

}

**return** set;

}

**public** **static** String insertCharacter(String s, **char** ch, **int** i) {

String begin = s.substring(0, i);

String end = s.substring(i);

**return** begin + ch + end;

}

}

/\*

\* OUTPUT

\*

\* Permutations of inputString(XYZ) are: [XYZ, XZY, YZX, ZYX, ZXY, YXZ]

\*

\*/

**public** **class** RemoveAllVowelsFromString {

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter the string...");

String inputString = sc.nextLine();

String newInputString = inputString.replaceAll("[AEIOUaeiou]", "");

System.***out***.println("The string without vowels...");

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

sc.close();

}

}

**public** **class** ReverseArray {

**static** **void** reverseArray(**int** inputArray[]) {

System.***out***.println(

"Array Before Reverse : " + Arrays.*toString*(inputArray));

**int** temp;

**for** (**int** i = 0, j = inputArray.length - 1; i < (inputArray.length / 2); i++, j--) {

temp = inputArray[i];

inputArray[i] = inputArray[j];

inputArray[j] = temp;

}

System.***out***.println(

"Array After Reverse : " + Arrays.*toString*(inputArray));

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

}

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

*reverseArray*(**new** **int**[] { 4, 5, 8, 9, 10 });

*reverseArray*(**new** **int**[] { 12, 9, 21, 17, 33, 7 });

*reverseArray*(**new** **int**[] { 891, 569, 921, 187, 343, 476, 555 });

}

}

**public** **class** ReverseNumberEx {

**public** **static** **void** main(String... args) {

**int** number = 12345; // number to be reversed

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

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

}

**public** **static** **int** reverseNumber(**int** number) {

**int** reverse = 0;

**int** remainder;

**while** (number > 0) {

remainder = number % 10;

number = number / 10;

reverse = reverse \* 10 + remainder;

}

**return** reverse;

}

}

/\*OUTPUT

Original number: 12345

Reversed number: 54321

\*/

**public** **class** ReverseStringEx {

**public** **static** **void** main(String... args) {

String originalString = "abcde"; // String to be reversed

System.***out***.println("Original String: " + originalString);

System.***out***.println(

"Reversed String: " + *reverseString*(originalString));

}

/\*

\* return reversed String.

\*/

**public** **static** String reverseString(String originalString) {

**char** ar[] = originalString.toCharArray();

**char** temp;

**for** (**int** i = 0, j = ar.length - 1; i < (ar.length / 2); i++, j--) {

temp = ar[i];

ar[i] = ar[j];

ar[j] = temp;

}

**return** **new** String(ar);

}

}

/\*OUTPUT

Original String: abcde

Reversed String: edcba

\*/

**public** **class** ReverseWordsInSentenceEx {

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

String inputString = "i am doing good";

System.***out***.println("Original String : "

+ inputString);

System.***out***.print("String after reversing words in sentence : ");

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

}

**public** **static** **char**[] reverseWordsInSentence(String inputString) {

**char** inputAr[] = inputString.toCharArray();

**char** outputAr[] = **new** **char**[inputString.length()];

**int** spacePos = 0, subStringLength = 0;

**int** outputArIndex = 0;

**for** (**int** inputArIndex = inputAr.length - 1; inputArIndex >= 0; inputArIndex--) { //strat iterating from last position

**if** (inputArIndex == 0) { // if we have reached starting index of inputAr.

**while** (!String.*valueOf*(inputAr[inputArIndex]).equals(" ")) { //start iterating from start, as soon as you find space exit while loop.

outputAr[outputArIndex++] = inputAr[inputArIndex++];

}

**return** outputAr;

}

**if** (String.*valueOf*(inputAr[inputArIndex]).equals(" ")) { //we have found space.

spacePos = inputArIndex; // store space position found in variable.

subStringLength = inputAr.length - 1 - spacePos; //calculate length of substring we have found

**while** (outputArIndex < subStringLength) {

outputAr[outputArIndex++] = inputAr[++spacePos];

}

outputAr[outputArIndex++] = ' '; //inserting space at last and then incrementing outputArIndex

}

}

**return** outputAr;

}

}

/\*

\* OUTPUT

\*

\* Original String : i am doing good String after reversing words in sentence :

\* good doing am i

\*/

**public** **class** SegregateOddEvenNumberSorting {

**static** **void** sorting(**int** a[]) {

System.***out***.println("Array Before sorting : " + Arrays.*toString*(a));

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

**int** i = 0, j = a.length - 1;

**while** (i < j) {

**while** (a[i] % 2 == 0)

i++;

**while** (a[j] % 2 != 0)

j--;

**if** (i < j) {

**int** temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

j--;

}

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

}

System.***out***.println("Array After sorting : " + Arrays.*toString*(a));

// **TODO**: two sub sets sorting goes

System.***out***.println("i=" + i + " , j=" + j);

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

}

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

*sorting*(**new** **int**[] { 4, 5, 8, 9, 10 });

*sorting*(**new** **int**[] { 12, 9, 21, 17, 33, 7 });

}

}

**public** **class** SegregateZerosAndOnesFromArray {

**static** **void** segregateZerosAndOnes(**int** a[]) {

System.***out***.println("Array Before segregate : " + Arrays.*toString*(a));

**int** i = 0, j = a.length - 1;

**while** (i < j) {

**while** (a[i] == 0)

i++;

**while** (a[j] == 1)

j--;

**if** (i < j) {

**int** temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

j--;

}

}

System.***out***.println("Array After segregate : " + Arrays.*toString*(a));

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

}

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

*segregateZerosAndOnes*(**new** **int**[] { 1, 0, 0, 1, 1 });

*segregateZerosAndOnes*(**new** **int**[] { 1, 1, 0, 0, 0, 1 });

}

}

**public** **class** SegregateZerosOnesAndTwosFromArray {

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

*segregateZerosOnesAndTwos*(**new** **int**[] { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 });

}

**static** **void** segregateZerosOnesAndTwos(**int** a[]) {

System.***out***.println("Array Before segregate : " + Arrays.*toString*(a));

**int** lo = 0;

**int** hi = a.length - 1;

**int** mid = 0, temp = 0;

**while** (mid <= hi) {

**switch** (a[mid]) {

**case** 0: {

temp = a[lo];

a[lo] = a[mid];

a[mid] = temp;

lo++;

mid++;

**break**;

}

**case** 1:

mid++;

**break**;

**case** 2:

temp = a[mid];

a[mid] = a[hi];

a[hi] = temp;

hi--;

**break**;

}

}

System.***out***.println("Array After segregate : " + Arrays.*toString*(a));

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

}

}

**public** **class** TwoConsecutiveNumbersWithHighestSum {

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

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

// int ar[] = { 1, 5, 8, 3, 6, 8, 7, 5, 4, 9 }; //Test array

// int ar[] = { 1, 2, 8, 3, 6, 1, 7, 5, 4, 7 }; //Test array

*findSetOfNumber*(ar);

}

**static** **void** findSetOfNumber(**int** ar[]) {

**int** sum = 0;

**int** maxSum = 0;

**int** temp1 = 0, temp2 = 0;

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

sum = ar[i - 1] + ar[i];

**if** (sum > maxSum) {

maxSum = sum;

temp1 = i - 1;

temp2 = i;

}

}

System.***out***.println("numbers= " + ar[temp1] + "," + ar[temp2] + " & index=" + temp1 + ", " + temp2);

}

}

/\*

\* OUTPUT

\*

\* numbers= 9,9 & index=5, 6

\*

\*/

**public** **class** ArmstrongNumberEx {

**public** **static** **void** main(String... args) {

**int** number = 153;

**int** sum = 0;

**int** temp = number; // temp will hold reference to number.

**while** (temp > 0) {

**int** rem = temp % 10;

sum += (rem \* rem \* rem);

temp = temp / 10;

}

**if** (number == sum)

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

**else**

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

}

}

/\*

\* OUTPUT

\*

\* 153 is armstrong number.

\*

\*/

**public** **class** DeleteRepetaedCharacterFromBothStringsEx {

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

String s1 = "aBc";

String s2 = "bcde";

System.***out***.println("s1 = " + s1 + " , s2 = " + s2);

**char** ar1[] = s1.toCharArray();

**char** ar2[] = s2.toCharArray();

**for** (**int** i = 0; i < ar1.length; i++) {

**boolean** bMatch = **false**;

**for** (**int** j = 0; j < ar2.length; j++) {

**if** ((String.*valueOf*(ar1[i])).toLowerCase().equals(

((String.*valueOf*(ar2[j])).toLowerCase()))) {

bMatch = **true**;

**break**;

}

}

**if** (bMatch) {

s1 = s1.replaceAll(String.*valueOf*(ar1[i]).toUpperCase(), "");

s1 = s1.replaceAll(String.*valueOf*(ar1[i]).toLowerCase(), "");

s2 = s2.replaceAll(String.*valueOf*(ar1[i]).toLowerCase(), "");

s2 = s2.replaceAll(String.*valueOf*(ar1[i]).toUpperCase(), "");

}

}

System.***out***.println("Modified s1 = " + s1 + ", Modified s2 = " + s2);

}

}

/\*

\* output

\*

\* s1 = abc , s2 = cde Modified s1 = ab, Modified s2 = de

\*

\*/

**public** **class** DividesWholeStringInStringsOfLengthOf3AndReturnList {

List<String> methodDividesStringAndReturnList(**int** n) {

String s = "abcdefghi";

// String s = "abcdefgh"; //Test String

// String s = "abcdefghij"; //Test String

**char** ch[] = s.toCharArray();

**int** ctr = 0;

List<String> l = **new** ArrayList<String>();

String temp = "";

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

**if** (ctr < n) {

temp += String.*valueOf*(ch[i]);

ctr++;

}

**if** (ctr == n || i == ch.length - 1) {

l.add(temp);

temp = "";

ctr = 0;

}

}

**return** l;

}

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

DividesWholeStringInStringsOfLengthOf3AndReturnList o = **new** DividesWholeStringInStringsOfLengthOf3AndReturnList();

List<String> l = o.methodDividesStringAndReturnList(3);

// Display list

Iterator<String> it = l.iterator();

**while** (it.hasNext()) {

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

}

}

}

/\*OUTPUT

abc

def

ghi

\*/

**public** **class** DividesWholeStringInStringsOfLengthOf3AndReturnListOfReversedStrings {

**static** List<String> split(String s) {

List<String> list = **new** ArrayList<String>();// to be returned, will keep

// reversed subString [i.e.

// cba and fed and ... ]

**char** ch[] = s.toCharArray();

**int** ctr = 0;

**char** chSmall[] = **new** **char**[3];

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

chSmall[ctr] = ch[i];

ctr++;

**if** (ctr == 3 || i + 1 == ch.length) {

ctr = 0;

*reverse*(chSmall);

list.add(**new** String(chSmall));

chSmall = **new** **char**[3];

}

}

**return** list;

}

**static** **void** reverse(**char** ch[]) {

**char** temp;

**for** (**int** i = 0, j = ch.length - 1; i <= ch.length / 2; i++, j--) {

temp = ch[i];

ch[i] = ch[j];

ch[j] = temp;

}

}

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

String s = "abcdefghi";

// String s = "abcdefgh"; //Test String

// String s = "abcdefghij"; //Test String

List<String> list = *split*(s);

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

}

}

/\*

\* OUTPUT

\*

\* [cba, fed, ihg]

\*

\*/

**public** **class** FactorialExample {

**public** **static** **void** main(String... args) {

**int** num = 4;

System.***out***.println(

"Factorial of " + num + " is: " + *findFactorail*(num));

}

/\*

\* return factorial of num.

\*/

**public** **static** **int** findFactorail(**int** num) {

**int** factorial = 1;

**while** (num > 0) {

factorial = factorial \* num;

num--;

}

**return** factorial;

}

}

/\*

\* OUTPUT

\*

\* Factorial of 4 is: 24

\*

\*/

**public** **class** FibonacciSeriesExampleInJava {

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

**int** n = 10; // number of elements in series.

*generateFibonacciSeries*(n);

}

**public** **static** **void** generateFibonacciSeries(**int** n) {

**int** first = 0; // first number of series

**int** second = 1; // second number of series

**int** temp;

System.***out***.print("FibonacciSeries: " + first + " " + second + " ");

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

temp = first + second;

first = second;

second = temp;

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

}

}

}

/\*OUTPUT

FibonacciSeries: 0 1 1 2 3 5 8 13 21 34 55 89

\*/

**public** **class** OccurrenceOfNumberInCubeEx {

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

**long** num = 1551;

**int** occurrenceOf = 1; // calculate OccurrenceOf this digit in number.

System.***out***.println("Cube of entered number(" + num + ") is: " + num \* num \* num);

System.***out***.println("Occurrence of " + occurrenceOf + " in cube of " + num + " is: "

+ *calculateOccurrenceOf*(num, occurrenceOf));

}

/\*\*

\* method return OccurrenceOf digit in cube of given number.

\*/

**public** **static** **int** calculateOccurrenceOf(**long** num, **int** occurrenceOf) {

**long** cubeOfNumber = num \* num \* num;

**int** count = 0;

**while** (cubeOfNumber > 0) {

**if** (cubeOfNumber % 10 == occurrenceOf) {

count++;

}

cubeOfNumber = cubeOfNumber / 10;

}

**return** count;

}

}

/\*OUTPUT

Cube of entered number(1551) is: 3731087151

Occurrence of 1 in cube of 1551 is: 3

\*/

**public** **class** PerfectNumberEx {

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

**int** n = 28;

System.***out***.println(

*isPerfectNumber*(n) == **true** ? n + " is perfect number."

: n + " is not a perfect number.");

}

**public** **static** **boolean** isPerfectNumber(**int** n) {

**int** sumOfDivisors = 1; // 1 is divisor of all the numbers, so initially

// keep the sumOfDivisors=1 and start check

// divisors from 2.

**for** (**int** i = 2; i <= n / 2; i++)

**if** (n % i == 0)

sumOfDivisors += i;

**if** (sumOfDivisors == n)

**return** **true**;

**else**

**return** **false**;

}

}

/\*

\* OUTPUT

\*

\* 28 is perfect number.

\*

\*/

**public** **class** PrimeNumberEx {

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

**int** n = 11;

System.***out***.println(*isPrimeNumber*(n) ? n + " is prime number." : n + " is not prime number.");

n = 12;

System.***out***.println(*isPrimeNumber*(n) ? n + " is prime number." : n + " is not prime number.");

n = 13;

System.***out***.println(*isPrimeNumber*(n) ? n + " is prime number." : n + " is not prime number.");

n = 14;

System.***out***.println(*isPrimeNumber*(n) ? n + " is prime number." : n + " is not prime number.");

}

/\*\*

\* returns true if number is prime.

\*/

**public** **static** **boolean** isPrimeNumber(**int** n) {

**for** (**int** i = 2; i <= Math.*sqrt*(n); i++) {

**if** (n % i == 0) {

**return** **false**;

}

}

**return** **true**; // means number wasn't divisible by any of the number, it's

// a prime number.

}

}

/\*OUTPUT

11 is prime number.

12 is not prime number.

13 is prime number.

14 is not prime number.

\*/

**public** **class** ReplaceAllVowelsIntoCapitalCaseEx {

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

String str = "java made so easy";

System.***out***.println("original string = " + str);

**char** c1[] = str.toCharArray();

**for** (**int** i = 0; i < c1.length; i++) {

**switch** (c1[i]) {

**case** 'a':

c1[i] = (**char**) (c1[i] - 32);

**break**;

**case** 'e':

c1[i] = (**char**) (c1[i] - 32);

**break**;

**case** 'i':

c1[i] = (**char**) (c1[i] - 32);

**break**;

**case** 'o':

c1[i] = (**char**) (c1[i] - 32);

**break**;

**case** 'u':

c1[i] = (**char**) (c1[i] - 32);

**break**;

}

}

str = **new** String(c1);

System.***out***.println(

"After replacing all vowels into capital case = " + str);

}

}

/\*

\* OUTPUT

\*

\* original string = java made so easy After replacing all vowels into capital

\* case = jAvA mAdE sO EAsy

\*

\*/

**public** **class** SumOfDigitsInNumberEx {

**public** **static** **void** main(String... args) {

**int** number = 1234;

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

System.***out***.println("sum of digits : " + *sumOfDigits*(number));

}

**public** **static** **int** sumOfDigits(**int** number) {

**int** sum = 0;

**int** remainder;

**while** (number > 0) {

remainder = number % 10;

number = number / 10;

sum += remainder;

}

**return** sum;

}

}

/\*OUTPUT

number : 1234

sum of digits : 10

\*/

**class** SwapNumbersWithoutThirdVariableEx {

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

**int** n1 = 7, n2 = 5;

System.***out***.println("before swapping, n1= " + n1 + " and n2= " + n2);

n1 = n1 + n2;

n2 = n1 - n2;

n1 = n1 - n2;

System.***out***.println("After swapping , n1= " + n1 + " and n2= " + n2);

}

}

/\*OUTPUT

before swapping, n1= 7 and n2= 5

After swapping , n1= 5 and n2= 7

\*/

**public** **class** CountConsecutiveElementInStrArray {

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

String inStr = "aaa bb cccc dd aa";

**char**[] ch = inStr.toCharArray();

Map<String, Integer> map = **new** LinkedHashMap<String, Integer>();

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

**int** count = 1;

String keyStr = String.*valueOf*(ch[i]);

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

**if** (ch[i] == ch[j]) {

keyStr = keyStr + String.*valueOf*(ch[j]);

count++;

i++;

} **else** {

**break**;

}

}

**if** (!keyStr.equals(" ")) {

map.put(keyStr, count);

}

}

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

}

}

**public** **class** FindDuplicateElementsFromStrArray {

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

String[] strArray = { "Java", "JSP", "Servlets", "Java", "Struts", "JSP", "JDBC" };

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

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

**if** (strArray[i].equals(strArray[j]) && i != j) {

System.***out***.println("Duplicate Element is : " + strArray[j]);

}

}

}

}

}

**public** **class** FindDuplicateInOneIterationEx {

**static** **int** *ar*[] = { 3, 7, 3, 5, 8, 9, 24, 8, 9 }; // given array

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

System.***out***.print("array : ");

**for** (**int** j = 0; j < *ar*.length; j++)

System.***out***.print(*ar*[j] + " "); // display it

*displayDuplicateInOneIteration*();

}

/\*

\* Method for displaying duplicate in one iteration

\*/

**public** **static** **void** displayDuplicateInOneIteration() {

**int** tempAr[] = **new** **int**[100];

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

**for** (**int** j = 0; j < *ar*.length; j++) {

**if** (tempAr[*ar*[j]] == 0) {

tempAr[*ar*[j]] = 1;

} **else** {

System.***out***.print(*ar*[j] + " ");

}

}

}

}

/\*

\* OUTPUT

\*

\* array : 3 7 3 5 8 9 24 8 9

\* duplicates : 3 8 9

\*

\*/

**public** **class** FindMissingNumberInSortedArrayEx {

**static** **int** *ar*[] = { 3, 5, 8, 44 }; // given array

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

*displayMissing*();

}

/\*

\* Method finds out missing number in given range.

\*/

**static** **public** **void** displayMissing() {

System.***out***.print("given array(already sorted): ");

**for** (**int** j = 0; j < *ar*.length; j++)

System.***out***.print(*ar*[j] + " "); // display it

System.***out***.print("\nNumbers missing between 1 to 100 in array : ");

**int** j = 0;

**for** (**int** i = 1; i <= 100; i++) {

**if** (j < *ar*.length && i == *ar*[j])

j++;

**else**

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

}

}

}

/\*

\* OUTPUT

\*

\* given array(already sorted): 3 5 8 44

\* Numbers missing between 1 to 100 in

\* array : 1 2 4 6 7 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

\* 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 45 46 47 48 49 50 51 52 53 54 55

\* 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81

\* 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

\*

\*/

**public** **class** FindOnlyDuplicateNumberInListEx {

**public** **static** **void** main(String... a) {

List<Integer> list = **new** ArrayList<Integer>();

**int** highestNumberInList = 10; // you may change it in future.

**for** (**int** i = 1; i <= highestNumberInList; i++) {

list.add(i);

}

list.add(5); // add duplicate number in list(please ensure you don't add

// more than one duplicate number in list)

System.***out***.println("list is: " + list);

System.***out***.println("Only duplicate number in list is: " + *findOnlyDuplicateNumberInList*(list, highestNumberInList));

}

/\*\*

\* returns only duplicate number in list.

\*/

**public** **static** **int** findOnlyDuplicateNumberInList(List<Integer> list, **int** highestNumberInList) {

**int** sumOfNumbersList = 0;

**for** (**int** n : list) {

sumOfNumbersList = sumOfNumbersList + n;

}

**int** onlyDuplicateNumberInList = sumOfNumbersList - ((highestNumberInList) \* (highestNumberInList + 1) / 2);

**return** onlyDuplicateNumberInList;

}

}

/\*

\* OUTPUT

\*

\* list is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 5]

\* Only duplicate number in list is: 5

\*

\*/

**public** **class** RemoveDuplicatesFromArrayList {

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

ArrayList<Object> al = **new** ArrayList<Object>();

al.add("java");

al.add('a');

al.add('b');

al.add('a');

al.add("java");

al.add(10.3);

al.add('c');

al.add(14);

al.add("java");

al.add(12);

System.***out***.println("Before Remove Duplicate elements:" + al);

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

**for** (**int** j = i + 1; j < al.size(); j++) {

**if** (al.get(i).equals(al.get(j))) {

al.remove(j);

j--;

}

}

}

System.***out***.println("After Removing duplicate elements:" + al);

}

}

**public** **class** RemoveDuplicatesFromCharcterArrayEx {

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

String str = "JavaJ2EE";

**char**[] ch = str.toCharArray();

System.***out***.println("Original Str: " + str);

System.***out***.println("Removed duplicates from str" + " : "

+ *removeDuplicates*(ch));

}

**public** **static** String removeDuplicates(**char**[] ch) {

**int** n = ch.length;

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

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

**if** (ch[i] == ch[j]) {

**int** temp = j;// set duplicate element index

// delete the duplicate element by shifting the elements to

// left

**for** (**int** k = temp; k < n - 1; k++) {

ch[k] = ch[k + 1];

}

j--;

n--;// reduce char array length

}

}

}

**return** **new** String(ch).substring(0, n);

}

}

**public** **class** RemoveDuplicatesFromSortedArrayEx {

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

// int[] duplicateSortedAr = { 1, 3, 7, 8, 8, 9, 14, 16, 16, 17, 17 };

**int**[] duplicateSortedAr = { 1, 3, 7, 8, 8, 9, 8, 9, 14, 16, 16, 17, 17 };

**int**[] nonDuplicateSortedAr = *removeDuplicates*(duplicateSortedAr);

System.***out***.print("Displaying contents of sorted array(with duplicate elements) : ");

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

System.***out***.print(duplicateSortedAr[i] + " ");

}

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

System.***out***.print("Displaying contents of sorted array(with non-duplicate elements) : ");

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

System.***out***.print(duplicateSortedAr[i] + " ");

}

}

/\*\*

\* This method returns sorted array with non duplicate elements

\*/

**public** **static** **int**[] removeDuplicates(**int**[] duplicateSortedAr) {

**int** i = 0, j = 1;

**if** (duplicateSortedAr.length < 2) { // means there is only one element

**return** duplicateSortedAr;

}

**while** (j < duplicateSortedAr.length) {

**if** (duplicateSortedAr[j] == duplicateSortedAr[i])

j++;

**else**

duplicateSortedAr[++i] = duplicateSortedAr[j++];

}

**int**[] nonDuplicateSortedAr = **new** **int**[i + 1];

**for** (**int** x = 0; x < nonDuplicateSortedAr.length; x++) {

nonDuplicateSortedAr[x] = duplicateSortedAr[x];

}

**return** nonDuplicateSortedAr;

}

}

/\*

\* OUTPUT

\*

\* Displaying contents of sorted array(with duplicate elements) : 1, 3, 7, 8, 8, 9, 14, 16, 16, 17, 17

\* Displaying contents of sorted array(with non-duplicate elements) : 1 3 7 8 9 14 16 17

\*

\*/

**public** **class** RemoveDuplicatesFromUnSortedArrayEx {

**static** **void** unique\_array(**int**[] a) {

System.***out***.println("Original Array : " + Arrays.*toString*(a));

// Assuming all elements in input array are unique

**int** n = a.length;

// Comparing each element with all other elements

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

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

// If any two elements are found equal

**if** (a[i] == a[j]) {

**int** temp = j;// set duplicate element index

// delete the duplicate element by shifting the elements to

// left

**for** (**int** k = temp; k < n - 1; k++) {

a[k] = a[k + 1];

}

j--;

n--;// reduce char array length

}

}

// System.out.println(Arrays.toString(a));

}

// Copying only unique elements of my\_array into array1

**int**[] array1 = Arrays.*copyOf*(a, n);

// Printing arrayWithoutDuplicates

System.***out***.println("Array with unique values : " + Arrays.*toString*(array1));

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

}

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

*unique\_array*(**new** **int**[] { 0, 3, -2, 4, 3, 2 });

*unique\_array*(**new** **int**[] { 10, 22, 10, 20, 11, 22 });

}

}

**public** **class** FindLongestSubstringWithoutRepeatingCharacters {

**static** **void** longestSubstring(String inputString) {

// Convert inputString to charArray

**char**[] charArray = inputString.toCharArray();

// Initialization

String longestSubstring = **null**;

**int** longestSubstringLength = 0;

// Creating LinkedHashMap with characters as keys and their position as

// values.

LinkedHashMap<Character, Integer> charPosMap = **new** LinkedHashMap<Character, Integer>();

// Iterating through charArray

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

**char** ch = charArray[i];

// If ch is not present in charPosMap, adding ch into charPosMap

// along with its position

**if** (!charPosMap.containsKey(ch)) {

charPosMap.put(ch, i);

}

// If ch is already present in charPosMap, reposioning the cursor i

// to the position of ch and clearing the charPosMap

**else** {

i = charPosMap.get(ch);

charPosMap.clear();

}

// Updating longestSubstring and longestSubstringLength

**if** (charPosMap.size() > longestSubstringLength) {

longestSubstringLength = charPosMap.size();

longestSubstring = charPosMap.keySet().toString();

}

}

System.***out***.println("Input String : " + inputString);

System.***out***.println("The longest substring : " + longestSubstring);

System.***out***.println("The longest Substring Length : "

+ longestSubstringLength);

}

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

*longestSubstring*("javaconceptoftheday");

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

*longestSubstring*("thelongestsubstring");

}

}

**public** **class** LongestPalindrom {

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

// System.out.println(longestPalindromeStr("1234"));

System.***out***.println(*longestPalindromeStr*("12321"));

// System.out.println(longestPalindromeStr("9912321456"));

// System.out.println(longestPalindromeStr("9912333321456"));

// System.out.println(longestPalindromeStr("12145445499"));

// System.out.println(longestPalindromeStr("1223213"));

// System.out.println(longestPalindromeStr("abb"));

}

// O(n^2)

**public** **static** String longestPalindromeStr(String str) {

**if** (str == **null**)

**return** **null**;

String longest = str.substring(0, 1);

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

// odd cases like 121

String palindrome = *intermediatePalindrome*(str, i, i);

**if** (palindrome.length() > longest.length()) {

longest = palindrome;

}

// even cases like 1221

palindrome = *intermediatePalindrome*(str, i, i + 1);

**if** (palindrome.length() > longest.length()) {

longest = palindrome;

}

}

**return** longest;

}

**public** **static** String intermediatePalindrome(String str, **int** left, **int** right) {

**if** (left > right)

**return** **null**;

**while** (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {

left--;

right++;

}

**return** str.substring(left + 1, right);

}

}

**public** **class** PalindromeEx {

**public** **static** **void** main(String... args) {

String inputString = "aabaa"; // String to be reversed

System.***out***.println(*isPalindrome*(inputString) ? inputString + " is a palindrome." : inputString + "is not a palindrome.");

}

/\*

\* returns true if inputString is palindrome.

\*/

**public** **static** **boolean** isPalindrome(String inputString) {

**char** ar[] = inputString.toCharArray();

**for** (**int** i = 0, j = ar.length - 1; i < (ar.length / 2); i++, j--) {

**if** (ar[i] != ar[j])

**return** **false**;

}

**return** **true**;

}

}

/\*

\* OUTPUT

\*

\* aabaa is a palindrome.

\*

\*/

**public** **class** PalindromeRecursionEx {

**public** **static** **void** main(String... args) {

String inputString = "aabaa";

System.***out***.println(*isPalindromeUsingRecursion*(inputString)

? inputString + " is a palindrome."

: inputString + "is not a palindrome.");

}

/\*\*

\* This methods finds out whether inputString is palindrome or not

\* recursively. Returns true if inputString is palindrome.

\*/

**public** **static** **boolean** isPalindromeUsingRecursion(String inputString) {

**if** (inputString.length() == 0 || inputString.length() == 1) {

**return** **true**;

}

**if** (inputString.charAt(0) == inputString

.charAt(inputString.length() - 1)) {

**return** *isPalindromeUsingRecursion*(

inputString.substring(1, inputString.length() - 1));

}

**return** **false**;

}

}

/\*

\* OUTPUT

\*

\* aabaa is a palindrome.

\*

\*/

**public** **class** PalindromeTillExample {

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

System.***out***.println("List of Palindrome between 1 to 1000");

**for** (**int** i = 10; i <= 1000; i++) {

**char**[] ar = String.*valueOf*(i).toCharArray();

**boolean** isPalindrome = **true**;

**for** (**int** j = 0; j < ar.length / 2; j++) {

**if** (ar[j] != ar[ar.length - 1 - j]) {

isPalindrome = **false**;

**break**;

}

}

**if** (isPalindrome)

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

}

}

}

/\*

List of Palindrome between 1 to 1000

11

22

33

44

55

66

77

88

99

101

111

121

131

141

151

161

171

181

191

202

212

222

232

242

252

262

272

282

292

303

313

323

333

343

353

363

373

383

393

404

414

424

434

444

454

464

474

484

494

505

515

525

535

545

555

565

575

585

595

606

616

626

636

646

656

666

676

686

696

707

717

727

737

747

757

767

777

787

**public** **class** RotationalPalindrome {

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

String inputStr = "aaaad";

*isRotationalPaliondrome*(inputStr);

}

**public** **static** **boolean** isRotationalPaliondrome(String inputStr) {

**char** ch[] = inputStr.toCharArray();

**for** (**int** x = 0; x < inputStr.length(); x++) {

*rotateString*(ch);

**if** (*isPalindrome*(ch)) {

System.***out***.println("inputString(i.e " + inputStr + ") is rotation of the palindrome: " + String.*valueOf*(ch));

**return** **true**;

}

}

**return** **false**;

}

**public** **static** **void** rotateString(**char**[] ch) {

**char** temp = ch[0];

**int** x = 0;

**for** (x = 0; x < ch.length - 1; x++) {

ch[x] = ch[x + 1];

}

ch[x] = temp;

}

**public** **static** **boolean** isPalindrome(**char** ch[]) {

**for** (**int** i = 0, j = ch.length - 1; i < (ch.length / 2); i++, j--) {

**if** (ch[i] != ch[j])

**return** **false**;

}

**return** **true**;

}

}

/\*\*

\* OUTPUT

\*

\* inputString(i.e aadaa) is rotation of the palindrome: aadaa

\*

\*/

**public** **class** FactorialRecursionEx {

**public** **static** **void** main(String... args) {

**int** num = 4;

System.***out***.println(

"Factorial of " + num + " is: " + *findFactorail*(num));

}

/\*

\* return factorial of num.

\*/

**public** **static** **int** findFactorail(**int** num) {

**if** (num == 0)

**return** 1;

**return** num \* *findFactorail*(num - 1);

}

}

/\*

\* OUTPUT

\*

\* Factorial of 4 is: 24

\*

\*/

**public** **class** FibonacciRecursionEx {

**public** **static** **void** main(String... args) {

**int** n = 10; // generate series upto n.

System.***out***.print("FibonacciSeries : 0 1 ");

**for** (**int** i = 2; i <= n; i++) {

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

}

}

/\*

\* return next number of FibonacciSeries using recursion.

\*/

**public** **static** **int** fibonacciRecursion(**int** n) {

**if** (n == 1 || n == 2) {

**return** 1;

}

**return** *fibonacciRecursion*(n - 1) + *fibonacciRecursion*(n - 2); // using

// tail

// recursion

}

}

/\*

\* OUTPUT

\*

\* FibonacciSeries : 0 1 1 2 3 5 8 13 21 34 55

\*

\*/

**public** **class** PalindromeRecursionEx {

**public** **static** **void** main(String... args) {

String inputString = "aabaa";

System.***out***.println(*isPalindromeUsingRecursion*(inputString)

? inputString + " is a palindrome."

: inputString + "is not a palindrome.");

}

/\*\*

\* This methods finds out whether inputString is palindrome or not

\* recursively. Returns true if inputString is palindrome.

\*/

**public** **static** **boolean** isPalindromeUsingRecursion(String inputString) {

**if** (inputString.length() == 0 || inputString.length() == 1) {

**return** **true**;

}

**if** (inputString.charAt(0) == inputString.charAt(inputString.length() - 1)) {

**return** *isPalindromeUsingRecursion*(inputString.substring(1, inputString.length() - 1));

}

**return** **false**;

}

}

/\*

\* OUTPUT

\*

\* aabaa is a palindrome.

\*

\*/

**public** **class** ReverseStringRecursionEx {

**public** **static** **void** main(String... args) {

String originalString = "abcde"; // String to be reversed

System.***out***.println("Original String: " + originalString);

System.***out***.print("Reversed String: ");

*reverseRecursively*(originalString);

}

/\*

\* reverse string recursively.

\*/

**public** **static** **void** reverseRecursively(String str) {

**if** (str.length() == 1) {

System.***out***.print(str);

} **else** {

*reverseRecursively*(str.substring(1, str.length()));

System.***out***.print(str.substring(0, 1));

}

}

}

/\*

\* OUTPUT

\*

\* Original String: abcde

\* Reversed String: edcba

\*

\*/