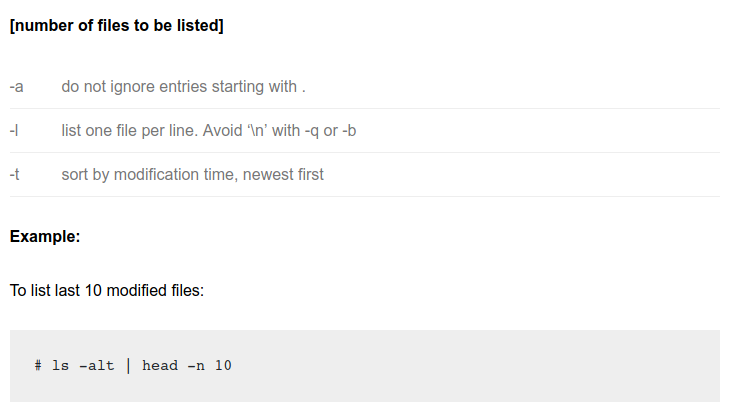
**Write a Unix command to give names of 10 last modified files in the current directory?**

****

**Design a fixed Thread Pool ExecutorService?**

**ExecutorService executor = Executors.newFixedThreadPool(10);**

First, we need to have a Runnable class, named WorkerThread.java

public class WorkerThread implements Runnable {

private String command;

public WorkerThread(String s){

this.command=s;

}

@Override

public void run() {

System.out.println(Thread.currentThread().getName()+" Start. Command = "+command);

processCommand();

System.out.println(Thread.currentThread().getName()+" End.");

}

private void processCommand() {

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

public String toString(){

return this.command;

}

}

Main Class

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class SimpleThreadPool {

public static void main(String[] args) {

**ExecutorService executor = Executors.newFixedThreadPool(10);**

for (int i = 0; i < 10; i++) {

Runnable worker = new WorkerThread("" + i);

executor.execute(worker);

}

executor.shutdown();

while (!executor.isTerminated()) {

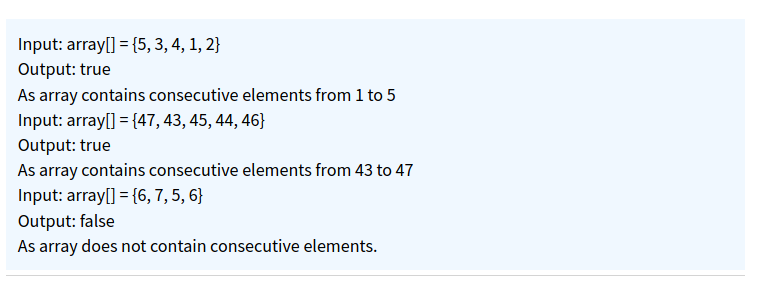
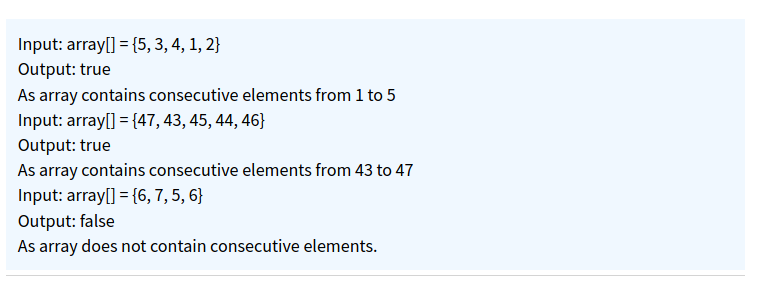
}

System.out.println("Finished all threads");

}

}

**Given an array, we need to check if array contains consecutive elements.**

****

Solution**:**

Simple solution will be to sort the array and check if elements are consecutive just by iterative over array but time complexity of this solution will be o(n^logn).

**public class ArrayConsecutiveElementMain{**

/\* Method return minimum value\*/

**private** **int** getMinimum(**int** arr[], **int** n)

{

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

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

**if** (arr[i] < min)

min = arr[i];

**return** min;

}

/\* Method return maximum value\*/

**private** **int** getMaximum(**int** arr[], **int** n)

{

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

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

**if** (arr[i] > max)

max = arr[i];

**return** max;

}

/\* This method checks if array elements are consecutive \*/

**public** **boolean** checkArrayContainsConsecutiveElements(**int** arr[], **int** n)

{

**if** ( n < 1 )

**return** **false**;

**int** min = getMinimum(arr, n);

**int** max = getMaximum(arr, n);

**if** (max - min + 1 == n)

{

**boolean**[] visited=**new** **boolean**[arr.length];

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

{

**if** ( visited[arr[i] - min] != **false** )

**return** **false**;

visited[arr[i] - min] = **true**;

}

**return** **true**;

}

**return** **false**;

}

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

{

ArrayConsecutiveElementMain acem=**new** ArrayConsecutiveElementMain();

**int** arr[]= {47, 43, 45, 44, 46};

**if**(acem.checkArrayContainsConsecutiveElements(arr, arr.length))

System.***out***.println(" Array elements are consecutive ");

**else**

System.***out***.println(" Array elements are not consecutive ");

**return**;

}

**}**

**What does wait() method do? Why it is always called inside a synchronised block?**

It can only be called from a synchronized block. It releases the lock on the object so that another thread can jump in and acquire a lock.

A wait() only makes sense when there is also a notify(), so it's always about communication between threads, and that needs synchronization to work correctly.

if we don't call wait() or notify() method from **synchronized context** we will receive **IllegalMonitorStateException** in Java. Race condition between wait() and notify() in Java which could exist if we don't call them inside synchronized method or block.

A wait() only makes sense when there is also a notify(), so it's always about communication between threads, and that needs synchronization to work correctly.

**What is recursion? Write the program for pre-order, Inorder and Post-Order traversal using recursion?**

Any function/method which calls itself is recursive. A recursive method solves a problem by calling a copy of itself to work on a smaller problem. This is called the recursive step. It is important to ensure that the recursion terminates. Hence we should also have a base case while having a recursive function.

**PreOrder Traversal using Recursion:**

private void preOrder(TreeNode node) {

if (node == null) {

return;

}

System.out.printf("%s ", node.data);

preOrder(node.left);

preOrder(node.right);

}

**InOrder Traversal using Recursion:**

private void inOrder(TreeNode node) {

if (node == null) {

return;

}

inOrder(node.left);

System.out.printf("%s ", node.data);

inOrder(node.right);

}

**Post-Order Traversal using Recursion:**

**private void postOrder(TreeNode node) {**

if (node == null) {

return;

}

postOrder(node.left);

postOrder(node.right);

System.out.printf("%s ", node.data);

}

**Count the frequency of each word in a very large file in lexicographically order?**

Lexicographical order is alphabetical order. We can use TreeMap to solve this problem.

**Remove the duplicate elements from an array without changing the order?**

LinkedHashSet can be used

Or else we can use HashSet with an Array.

Or a Nested loops wiht O(n^2)

**What is Right Join?**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

**Find k missing numbers from an array with duplicates?**

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

// given input

**int**[] input = { 1, 1, 2, 3, 5, 5, 7, 9, 9, 9 };

// let's create another array with same length

// by default all index will contain zero

// default value for int variable

**int**[] register = **new** **int**[input.length];

// now let's iterate over given array to

// mark all present numbers in our register

// array

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

register[i] = 1;

}

// now, let's print all the absentees

System.***out***.println("missing numbers in given array");

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

**if** (register[i] == 0) {

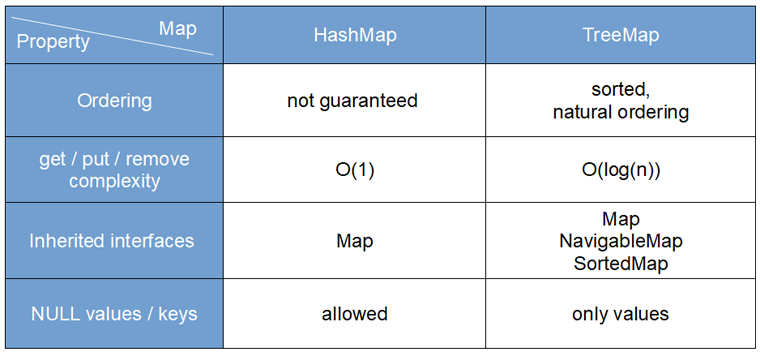
System.***out***.println(i);

}

}

**}**

**HashMap vs TreeMap**

****