**Basic of MultiThreading**

**1. Write a programe do to demonstrate the use of volatile keyword.**

**Code:-**

**package Voletile;**

**import java.util.Scanner;**

**class Test extends Thread {**

**private volatile boolean running = true;**

**public void run() {**

**while (running) {**

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

**}**

**}**

**public void shutdown() {**

**running = false;**

**}**

**}**

**class Main {**

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

**Test obj = new Test();**

**obj.start();**

**System.*out*.println("press enter to stop....");**

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

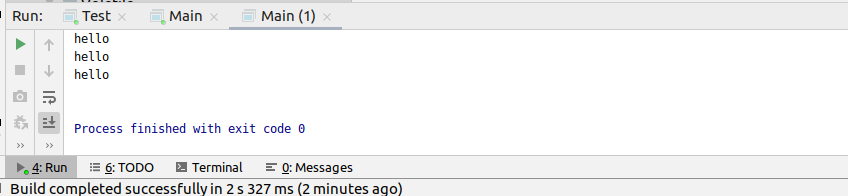
**input.nextLine();**

**obj.shutdown();**

**}**

**}**

**Output:-**

****

**2.Write a program to create a thread using Thread class and Runnable interface each.**

**Code:-**

**package demo;**

**class Hi1 implements Runnable{**

**public void run() {**

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

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

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**class Hello1 implements Runnable{**

**public void run() {**

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

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

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**public class ThreadDemo1 {**

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

**Hi1 obj1 = new Hi1();**

**Hello1 obj2 = new Hello1();**

**Thread t1 = new Thread(obj1);**

**Thread t2 = new Thread(obj2);**

**t1.start();**

**System.*out*.println(t1.isAlive());**

**try {**

**Thread.*sleep*(500);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**t2.start();**

**t1.join();**

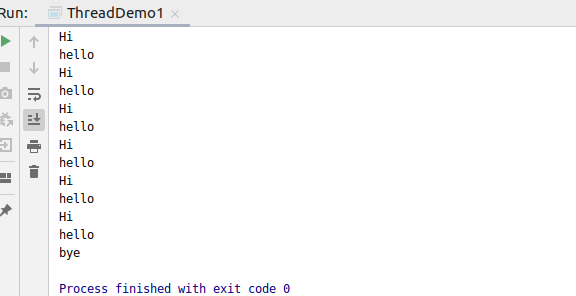
**t2.join();**

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

**}**

**}**

**Output:-**

****

**3.Write a program using synchronization block and synchronization method**

**Code:-**

**package sync;**

**class Counter**

**{**

**int count =0;**

**public synchronized void increment()**

**{**

**count++;**

**}**

**}**

**public class SyncDemo {**

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

**{**

**Counter c = new Counter();**

***// c.increment();***

**Thread t1 = new Thread(new Runnable()**

**{**

**public void run()**

**{**

**for(int i=0;i<1000;i++)**

**{**

**c.increment();**

**}**

**}**

**});**

**Thread t2 = new Thread(new Runnable()**

**{**

**public void run()**

**{**

**for(int i=0;i<1000;i++)**

**{**

**c.increment();**

**}**

**}**

**});**

**t1.start();**

**t2.start();**

**try {**

**t1.join();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**try {**

**t2.join();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

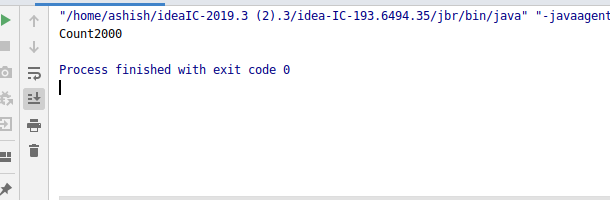
**}**

**System.*out*.println("Count" + c.count);**

**}**

**}**

**Output:-**

****

**4.Write a program to create a Thread pool of 2 threads where one Thread will print even numbers and other will print odd numbers.**

**Code:-**

**package ThreadPool;**

**public class OddEvenPrintMain {**

**boolean odd;**

**int count = 1;**

**int MAX = 20;**

**public void printOdd() {**

**synchronized (this) {**

**while (count < MAX) {**

**System.*out*.println("Checking odd loop");**

**while (!odd) {**

**try {**

**System.*out*.println("Odd waiting : " + count);**

**wait();**

**System.*out*.println("Notified odd :" + count);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**System.*out*.println("Odd Thread :" + count);**

**count++;**

**odd = false;**

**notify();**

**}**

**}**

**}**

**public void printEven() {**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e1) {**

**e1.printStackTrace();**

**}**

**synchronized (this) {**

**while (count < MAX) {**

**System.*out*.println("Checking even loop");**

**while (odd) {**

**try {**

**System.*out*.println("Even waiting: " + count);**

**wait();**

**System.*out*.println("Notified even:" + count);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**System.*out*.println("Even thread :" + count);**

**count++;**

**odd = true;**

**notify();**

**}**

**}**

**}**

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

**OddEvenPrintMain oep = new OddEvenPrintMain();**

**oep.odd = true;**

**Thread t1 = new Thread(new Runnable() {**

**@Override**

**public void run() {**

**oep.printEven();**

**}**

**});**

**Thread t2 = new Thread(new Runnable() {**

**@Override**

**public void run() {**

**oep.printOdd();**

**}**

**});**

**t1.start();**

**t2.start();**

**try {**

**t1.join();**

**t2.join();**

**} catch (InterruptedException e) {**

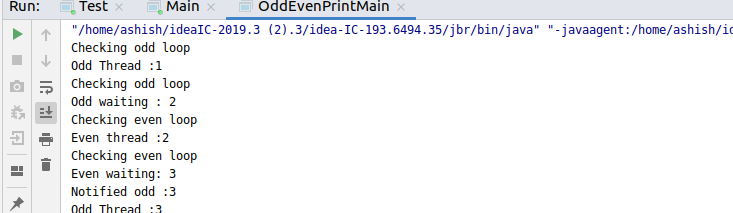
**e.printStackTrace();**

**}**

**}**

**}**

**Output:-**

****

**5.Write a program to demonstrate wait and notify methods.**

**Code:-**

**package WaitNotify;**

**public class WaitDemo {**

**int i = 10;**

**int display() {**

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

**i++;**

**return i;**

**}**

**}**

**class ClassDemo1 extends Thread {**

**private WaitDemo wd = new WaitDemo();**

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

**ClassDemo1 cd1 = new ClassDemo1();**

**ClassDemo1 cd2 = new ClassDemo1();**

**cd1.setName("Ashish");**

**cd2.setName("Ayush");**

**cd1.start();**

**cd2.start();**

**}**

**synchronized void display() {**

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

**notifyAll();**

**}**

**public void run() {**

**synchronized (this) {**

**try {**

**{**

**notify();**

**System.*out*.println("The thread is " + *currentThread*().getName());**

**wait();**

**System.*out*.println("The value is " + wd.display());**

**}**

**} catch (InterruptedException e) {**

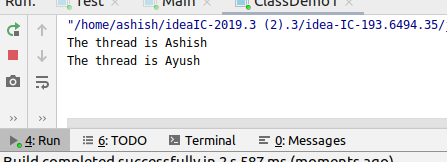
**}**

**}**

**}**

**}**

**Output:-**

****

**6.Write a program to demonstrate sleep and join methods.**

**Code:-**

**package demo;**

**class Hi1 implements Runnable{**

**public void run() {**

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

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

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**class Hello1 implements Runnable{**

**public void run() {**

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

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

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**public class ThreadDemo1 {**

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

**Hi1 obj1 = new Hi1();**

**Hello1 obj2 = new Hello1();**

**Thread t1 = new Thread(obj1);**

**Thread t2 = new Thread(obj2);**

**t1.start();**

**System.*out*.println(t1.isAlive());**

**try {**

**Thread.*sleep*(500);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**t2.start();**

**t1.join();**

**t2.join();**

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

**}**

**}**

**7.Run a task with the help of callable and store it's result in the Future.**

**Code:-**

**import java.io.IOException;**

|  |
| --- |
| **import java.util.Random;** |
|  |

|  |
| --- |
| **import java.util.concurrent.\*;** |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **public class CallableFuture** |
|  |

|  |
| --- |
| **{** |
|  |

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

|  |
| --- |
| **ExecutorService executer = Executors.newCachedThreadPool();** |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **Future<Integer> future = executer.submit(new Callable<Integer>()** |
|  |

|  |
| --- |
| **{** |
|  |

|  |
| --- |
| **@Override** |
|  |

|  |
| --- |
| **public Integer call() throws Exception {** |
|  |

|  |
| --- |
| **Random random = new Random();** |
|  |

|  |
| --- |
| **int duration = random.nextInt(4000);** |
|  |

|  |
| --- |
| **if(duration>2000)** |
|  |

|  |
| --- |
| **throw new IOException("Sleeping for too long.");** |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| **try {** |
|  |

|  |
| --- |
| **Thread.sleep(duration); }** |
|  |

|  |
| --- |
| **catch(InterruptedException e)** |
|  |

|  |
| --- |
| **{** |
|  |

|  |
| --- |
| **e.printStackTrace();** |
|  |

|  |
| --- |
| **}** |
|  |

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

|  |
| --- |
| **return duration;** |
|  |

|  |
| --- |
| **}** |
|  |

|  |
| --- |
| **});** |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **executer.shutdown();** |
|  |

|  |
| --- |
| **try {** |
|  |

|  |
| --- |
| **System.out.println("Result is: " + future.get());** |
|  |

|  |
| --- |
| **}** |
|  |

|  |
| --- |
| **catch (InterruptedException e)** |
|  |

|  |
| --- |
| **{ e.printStackTrace(); }** |
|  |

|  |
| --- |
| **catch (ExecutionException e)** |
|  |

|  |
| --- |
| **{ IOException ex = (IOException) e.getCause();** |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| **System.out.println(ex.getMessage());** |
|  |

|  |
| --- |
| **}** |
|  |

|  |
| --- |
| **}** |
|  |

**}**

**8.Write a program to demonstrate the use of semaphore**

**Code:-**

**package Semaphore;**

**import java.util.concurrent.\*;**

**class Resouce**

**{**

**static int *count* = 0;**

**}**

**class MyDemo extends Thread**

**{**

**Semaphore sem;**

**String threadName;**

**public MyDemo(Semaphore sem, String threadName)**

**{**

**super(threadName);**

**this.sem = sem;**

**this.threadName = threadName;**

**}**

**@Override**

**public void run() {**

**if(this.getName().equals("X"))**

**{**

**System.*out*.println("Starting " + threadName);**

**try**

**{**

**System.*out*.println(threadName + " waiting for a permit.");**

**sem.acquire();**

**System.*out*.println(threadName + " gets a permit.");**

**for(int i=0; i < 7; i++)**

**{**

**Resouce.*count*++;**

**System.*out*.println(threadName + ": " + Resouce.*count*);**

**Thread.*sleep*(20);**

**}**

**} catch (InterruptedException exc) {**

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

**}**

**System.*out*.println(threadName + " releases the permit.");**

**sem.release();**

**}**

**else**

**{**

**System.*out*.println("Starting " + threadName);**

**try**

**{**

**System.*out*.println(threadName + " waiting for a permit.");**

**sem.acquire();**

**System.*out*.println(threadName + " gets a permit.");**

**for(int i=0; i < 7; i++)**

**{**

**Resouce.*count*--;**

**System.*out*.println(threadName + ": " + Resouce.*count*);**

**Thread.*sleep*(20);**

**}**

**} catch (InterruptedException exc) {**

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

**}**

**System.*out*.println(threadName + " releases the permit.");**

**sem.release();**

**}**

**}**

**private Object getName() {**

**}**

**public void join() {**

**}**

**public void start() {**

**}**

**}**

**class SemTest**

**{**

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

**{**

**Semaphore sem = new Semaphore(1);**

**MyDemo md1 = new MyDemo(sem, "X");**

**MyDemo md2 = new MyDemo(sem, "Y");**

**md1.start();**

**md2.start();**

**md1.join();**

**md2.join();**

**System.*out*.println("count: " + Resouce.*count*);**

**}**

**}**

**9.Write a program to demonstrate the use of CountDownLatch**

**Code:-**

**package countdlatch;**

**import java.util.concurrent.CountDownLatch;**

**import java.util.concurrent.ExecutorService;**

**import java.util.concurrent.Executors;**

**import java.util.concurrent.TimeUnit;**

**class Processor implements Runnable{**

**private CountDownLatch latch;**

**public Processor(CountDownLatch latch)**

**{**

**this.latch = latch;**

**}**

**public void run()**

**{**

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

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**latch.countDown();**

**}**

**}**

**public class App {**

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

**CountDownLatch latch = new CountDownLatch(3);**

**ExecutorService executor = Executors.*newFixedThreadPool*(3);**

**for(int i=0;i<3;i++)**

**{**

**executor.submit(new Processor(latch));**

**}**

**try {**

**latch.await();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

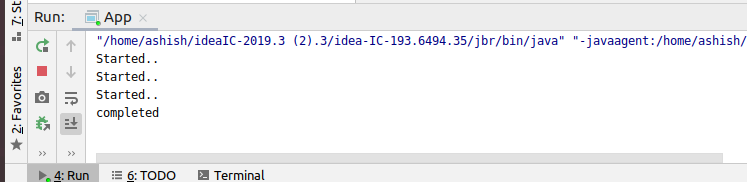
**}**

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

**}**

**}**

**Output:-**

****

**10.Write a program which creates deadlock between 2 threads**

**Code:-**

**public class Deadlock {**

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

**final String resource1 = "ashish";**

**final String resource2 = "ayush";**

***// t1 tries to lock resource1 then resource2***

**Thread t1 = new Thread() {**

**public void run() {**

**synchronized (resource1) {**

**System.*out*.println("Thread 1: locked resource 1");**

**try { Thread.*sleep*(100);} catch (Exception e) {}**

**synchronized (resource2) {**

**System.*out*.println("Thread 1: locked resource 2");**

**}**

**}**

**}**

**};**

***// t2 tries to lock resource2 then resource1***

**Thread t2 = new Thread() {**

**public void run() {**

**synchronized (resource2) {**

**System.*out*.println("Thread 2: locked resource 2");**

**try { Thread.*sleep*(100);} catch (Exception e) {}**

**synchronized (resource1) {**

**System.*out*.println("Thread 2: locked resource 1");**

**}**

**}**

**}**

**};**

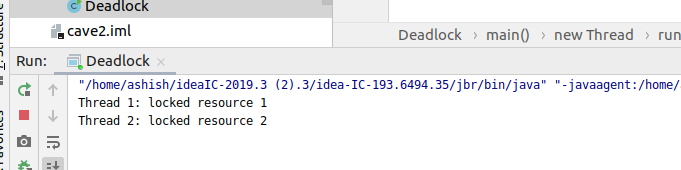
**t1.start();**

**t2.start();**

**}**

**}**

**Output:-**

****