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

import java.util.Scanner;

class Processor extends Thread {

private volatile boolean running = true;

public void run() {

while(running) {

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

try {

Thread.*sleep*(50);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public void shutdown() {

running = false;

System.*out*.println("Termination button pressed");

}

}

public class q1\_volatile {

public static void main(String[] args) {

Processor pro = new Processor();

pro.start();

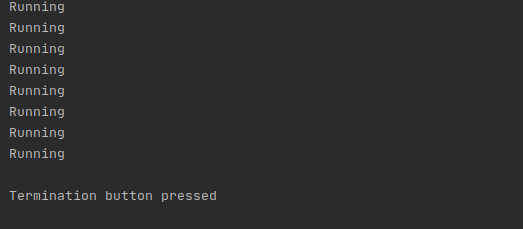
// Wait for the enter key

new Scanner(System.*in*).nextLine();

pro.shutdown();

}

}



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

//Thread creation via extending Thread class

class Runner extends Thread {

@Override

public void run() {

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

System.*out*.println("Hello: " + i);

try {

Thread.*sleep*(100);

} catch (InterruptedException e) {

// *TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

}

public class q1\_ThreadCreation {

public static void main(String[] args) {

Runner runner1 = new Runner();

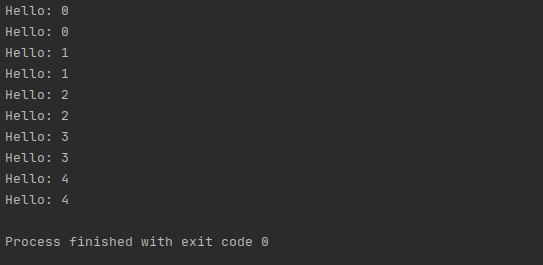
runner1.start();

Runner runner2 = new Runner();

runner2.start();

}

}



//Thread creation via Implementing Runnable Interface

class Runner2 implements Runnable {

@Override

public void run() {

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

System.*out*.println("Hello: " + i);

try {

Thread.*sleep*(100);

} catch (InterruptedException e) {

// *TODO Auto-generated catch block*

e.printStackTrace();

}

}

}

}

public class q1\_2\_ThreadCreation {

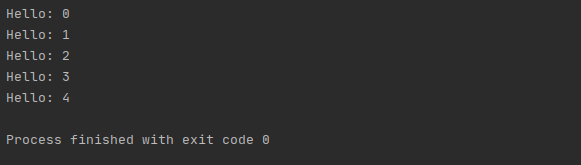
public static void main(String[] args) {

Thread thread1 = new Thread(new Runner());

thread1.start();

}

}



1. Write a program using synchronization block and synchronization method

//synchronization method

import java.util.List;

class Line{

//getLine method is Synchronize so no two thread can acess it same time

synchronized public void getLine(){

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

System.*out*.println(Thread.*currentThread*().getName()+" Cycle :"+i);

try {

Thread.*sleep*(400);

}

catch (Exception e){

System.*out*.println(e);

}

}

}

}

class Train extends Thread

{

Line line;

Train(Line line){

this.line=line;

}

public void run()

{

line.getLine();

}

}

class q3{

public static void main(String[] args) {

Line obj=new Line();

Train Train1=new Train(obj);

Train Train2=new Train(obj);

Train Train3=new Train(obj);

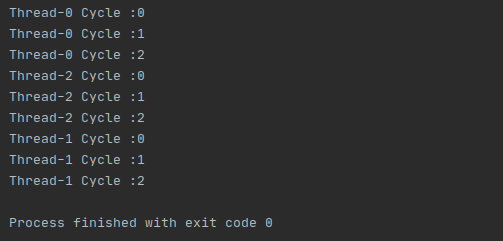
Train1.start();

Train2.start();

Train3.start();

}

}



//Synchronization block

import java.io.\*;

import java.util.\*;

class Block

{

String name = "";

public int count = 0;

public void setName(String geek, List<String> list)

{

synchronized(this)

{

name = "Anonymous";

count++; // how many threads change geek's name.

}

System.*out*.println(count);

list.add(geek);

}

}

class q3\_2

{

public static void main (String[] args)

{

Block bk = new Block();

List<String> list = new ArrayList<String>();

bk.setName("mohit", list);

bk.setName("Shivam", list);

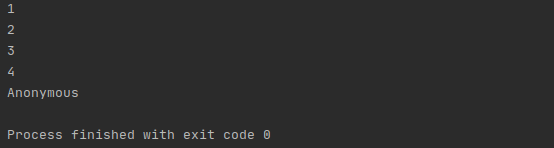
bk.setName("Vishal", list);

bk.setName("Shubham", list);

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

}

}



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

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Scanner;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

// Task class to be executed (Step 1)

class Task implements Runnable {

private String name;

public Task(String s) {

name = s;

}

// Prints task name and sleeps for 1s

// This Whole process is repeated 5 times

public void run() {

try {

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

if (i == 0) {

Date d = new Date();

SimpleDateFormat ft = new SimpleDateFormat("hh:mm:ss");

System.*out*.println("Initialization Time for"

+ " task name - " + name + " = " + ft.format(d));

//prints the initialization time for every task

} else {

Date d = new Date();

SimpleDateFormat ft = new SimpleDateFormat("hh:mm:ss");

System.*out*.println("Executing Time for task name - " +

name + " = " + ft.format(d));

// prints the execution time for every task

}

Thread.*sleep*(1000);

}

System.*out*.println(name + " complete");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

class q4 {

public static void main(String[] args) {

// Maximum number of threads in thread pool

System.*out*.println("Enter number of threads in thread pool");

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

final int MAX\_T = scanner.nextInt();

// creates five tasks

Runnable r1 = new Task("task 1");

Runnable r2 = new Task("task 2");

Runnable r3 = new Task("task 3");

Runnable r4 = new Task("task 4");

Runnable r5 = new Task("task 5");

ExecutorService pool = Executors.*newFixedThreadPool*(MAX\_T);

pool.execute(r1);

pool.execute(r2);

pool.execute(r3);

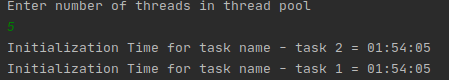
pool.execute(r4);

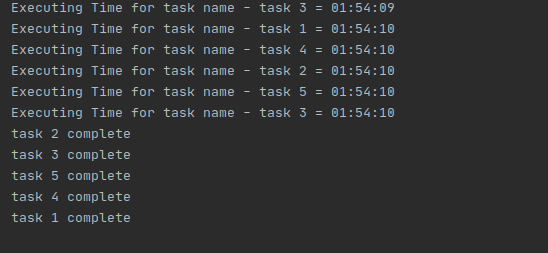
pool.execute(r5);

pool.shutdown();

}

}





1. Write a program to demonstrate wait and notify methods.

import java.util.Scanner;

class Processor2 {

public void produce() throws InterruptedException {

synchronized (this) {

System.*out*.println("Producer thread running ....");

wait();

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

}

}

public void consume() throws InterruptedException {

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

Thread.*sleep*(2000);

synchronized (this) {

System.*out*.println("Waiting for return key.");

scanner.nextLine();

System.*out*.println("Return key pressed.");

notify();

Thread.*sleep*(5000);

}

}

}

public class q5\_wait\_notify {

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

Processor2 p2 = new Processor2();

Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

try {

p2.produce();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

Thread t2 = new Thread(new Runnable() {

@Override

public void run() {

try {

p2.consume();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

t1.start();

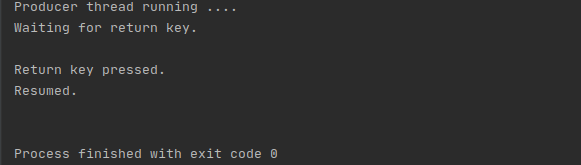
t2.start();

t1.join();

t2.join();

}

}



1. Write a program to demonstrate sleep and join methods.

// Java program to demonstrate inter-thread communication

// (wait(), join() and

import java.util.Scanner;

public class q6

{

public static void main(String[] args)

throws InterruptedException

{

final PC pc = new PC();

Thread t1 = new Thread(new Runnable()

{

@Override

public void run()

{

try

{

pc.produce();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

Thread t2 = new Thread(new Runnable()

{

@Override

public void run()

{

try

{

pc.consume();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

}

});

t1.start();

t2.start();

t1.join();

t2.join();

}

public static class PC

{

public void produce()throws InterruptedException

{

synchronized(this)

{

System.*out*.println("producer thread running");

wait();

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

}

}

public void consume()throws InterruptedException

{

Thread.*sleep*(1000);

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

synchronized(this)

{

System.*out*.println("Waiting for return key.");

s.nextLine();

System.*out*.println("Return key pressed");

notify();

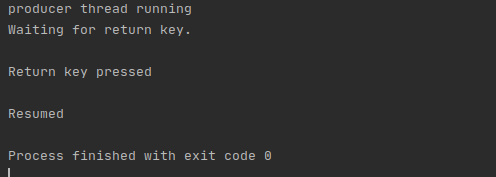
Thread.*sleep*(2000);

}

}

}

}



1. Run a task with the help of callable and store it's result in the Future.
2. Write a program to demonstrate the use of semaphore

import java.util.concurrent.\*;

class Shared

{

static int *count* = 0;

}

class MyThread extends Thread

{

Semaphore sem;

String threadName;

public MyThread(Semaphore sem, String threadName)

{

super(threadName);

this.sem = sem;

this.threadName = threadName;

}

@Override

public void run() {

// run by thread A

if(this.getName().equals("A"))

{

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

try

{

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

sem.acquire();

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

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

{

Shared.*count*++;

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

// Now, allowing a context switch -- if possible.

// for thread B to execute

Thread.*sleep*(10);

}

} catch (InterruptedException exc) {

System.*out*.println(exc);

}

// Release the permit.

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

sem.release();

}

// run by thread B

else

{

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

try

{

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

sem.acquire();

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

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

{

Shared.*count*--;

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

// Now, allowing a context switch -- if possible.

// for thread A to execute

Thread.*sleep*(10);

}

} catch (InterruptedException exc) {

System.*out*.println(exc);

}

// Release the permit.

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

sem.release();

}

}

}

public class q8Semaphore

{

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

{

Semaphore sem = new Semaphore(2);

MyThread mt1 = new MyThread(sem, "A");

MyThread mt2 = new MyThread(sem, "B");

MyThread mt3 = new MyThread(sem, "C");

MyThread mt4 = new MyThread(sem, "D");

// stating threads A and B

mt1.start();

mt2.start();

mt3.start();

mt4.start();

mt1.join();

mt2.join();

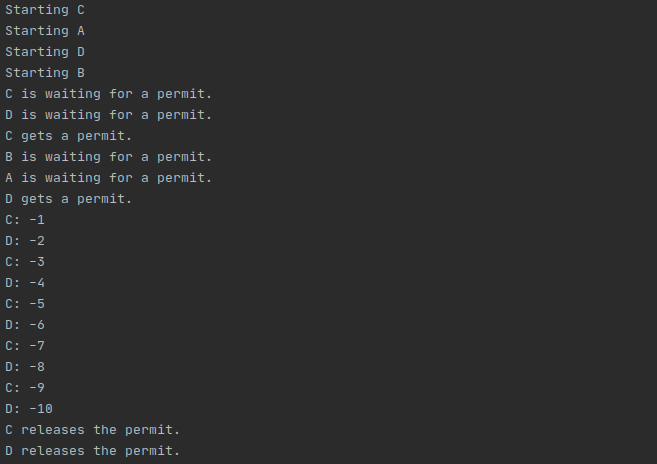
mt3.join();

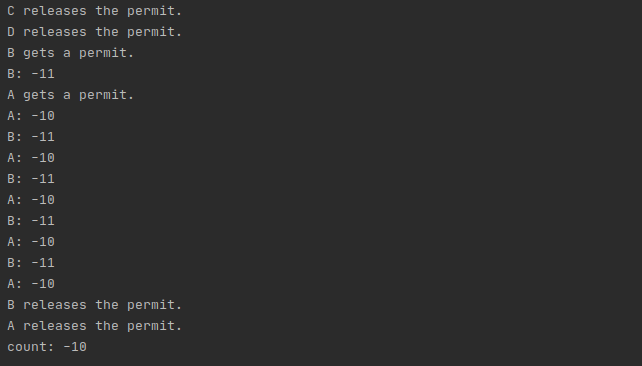
mt4.join();

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

}

}





1. Write a program to demonstrate the use of CountDownLatch

import java.util.Scanner;

import java.util.concurrent.CountDownLatch;

public class q9CountDownLatch{

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

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

System.*out*.println("Enter no of threads to wait for(1-5)...........");

int no\_of\_threads=sc.nextInt();

CountDownLatch countDownLatch=new CountDownLatch(no\_of\_threads);

coders coder1=new coders(1000,countDownLatch,"Shivam");

coders coder2=new coders(2000,countDownLatch,"vishal");

coders coder3=new coders(3000,countDownLatch,"mohit");

coders coder4=new coders(3500,countDownLatch,"saksham");

coders coder5=new coders(4000,countDownLatch,"shreyash");

coder1.start();

coder2.start();

coder3.start();

coder4.start();

coder5.start();

//Latch waits for given no of threads

countDownLatch.await();

System.*out*.println(Thread.*currentThread*().getName()+": has finished");

}

}

class coders extends Thread {

private int delay;

CountDownLatch latch;

public coders(int delay, CountDownLatch latch, String name) {

super(name);

this.delay = delay;

this.latch = latch;

}

public void run() {

try {

Thread.*sleep*(delay);

latch.countDown();

System.*out*.println("Current thread:"+"->" + Thread.*currentThread*().getName() + " Finished");

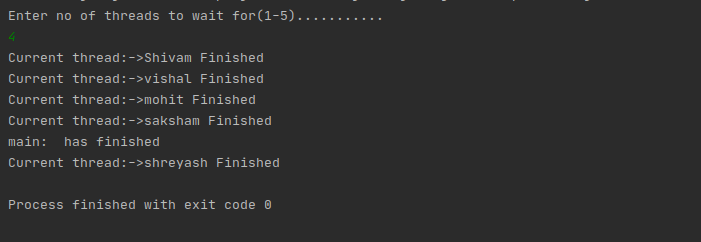
} catch (InterruptedException e) {

e.printStackTrace();

}

}

}



1. Write a program which creates deadlock between 2 threads

class Util

{

static void sleep(long millis)

{

try

{

Thread.*sleep*(millis);

}

catch (InterruptedException e)

{

e.printStackTrace();

}

}

}

// shared with both threads

class Shared

{

// first synchronized method

synchronized void test1(Shared s2)

{

System.*out*.println("test1-begin");

Util.*sleep*(1000);

// taking object lock of s2 enters

// into test2 method

s2.test2(this);

System.*out*.println("test1-end");

}

// second synchronized method

synchronized void test2(Shared s1)

{

System.*out*.println("test2-begin");

Util.*sleep*(1000);

s1.test1(this);

System.*out*.println("test2-end");

}

}

class Thread1 extends Thread

{

private Shared s1;

private Shared s2;

public Thread1(Shared s1, Shared s2)

{

this.s1 = s1;

this.s2 = s2;

}

@Override

public void run()

{

// taking object lock of s1 enters

// into test1 method

s1.test1(s2);

}

}

class Thread2 extends Thread

{

private Shared s1;

private Shared s2;

public Thread2(Shared s1, Shared s2)

{

this.s1 = s1;

this.s2 = s2;

}

@Override

public void run()

{

s2.test2(s1);

}

}

public class q10\_Deadlock

{

public static void main(String[] args)

{

// creating one object

Shared s1 = new Shared();

// creating second object

Shared s2 = new Shared();

// creating first thread and starting it

Thread1 t1 = new Thread1(s1, s2);

t1.start();

// creating second thread and starting it

Thread2 t2 = new Thread2(s1, s2);

t2.start();

// sleeping main thread

Util.*sleep*(2000);

}

}

