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

package dayThree;

public class TestVolatile extends Thread{

//Without volatile the thread forever hangs and doesnt end.

volatile boolean keepRunning = true;

public void run(){

long count = 0;

while(keepRunning){

count++;

}

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

}

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

TestVolatile t = new TestVolatile();

t.start();

Thread.*sleep*(100);

System.*out*.println("After sleeping in main");

t.keepRunning = false;

t.join();

System.*out*.println("keepRunning "+ t.keepRunning);

}

}

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

THREAD CLASS

package dayThree.Q2;

public class CreateUsingThread extends Thread {

public void run(){

System.*out*.println("Run method executed by thread");

}

public static void main(String[] args) {

CreateUsingThread t = new CreateUsingThread();

t.start();

System.*out*.println("Main method executed");

}

}

RUNNABLE

package dayThree.Q2;

class Abc{

public static void m1() {

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

}

}

public class CreatedUsingRunnable extends Abc implements Runnable{

public void run(){

System.*out*.println("Run method executed");

}

public static void main(String[] args) {

CreatedUsingRunnable t = new CreatedUsingRunnable();

t.*m1*();

Thread t1 = new Thread(t);

t1.start();

System.*out*.println("Main method executed");

}

}

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

SYNCHRO BLOCK

package dayThree.Q3;

class SynchoBlock {

void printTable(int n){

synchronized(this){//synchronized block

for(int i=1;i<=5;i++){

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

try{

Thread.*sleep*(400);

}catch(Exception e){System.*out*.println(e);}

}

}

}//end of the method

}

class TestSynchronizedBlock2{

public static void main(String args[]){

final SynchoBlock obj = new SynchoBlock();//only one object

Thread t1=new Thread(){

public void run(){

System.*out*.print("Th1 : ");

obj.printTable(5);

}

};

Thread t2=new Thread(){

public void run(){

System.*out*.print("Th2 : ");

obj.printTable(100);

}

};

t1.start();

t2.start();

}

}

SYNCHRO METHOD

package dayThree.Q3;

class Line

{

// if multiple threads(trains) will try to

// access this unsynchronized method,

// they all will get it. So there is chance

// that Object's state will be corrupted.

public void SynchroMethod()

{

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

{

System.*out*.println(i);

try

{

Thread.*sleep*(0);

}

catch (Exception e)

{

System.*out*.println(e);

}

}

}

}

class Train extends Thread

{

// reference to Line's Object.

Line line;

Train(Line line)

{

this.line = line;

}

@Override

public void run()

{

line.SynchroMethod();

}

}

class Test {

public static void main(String[] args)

{

// Object of Line class that is shared

// among the threads.

Line obj = new Line();

// creating the threads that are sharing the same Object.

Train train1 = new Train(obj);

Train train2 = new Train(obj);

// threads start their execution.

train1.start();

train2.start();

}

}

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.**

package dayThree.Q4;

public class ThreadPoolOddEven {

public static void main(String args[]) {

Printer print = new Printer();

Thread t1 = new Thread(new TaskEvenOdd(print, 10, false));

Thread t2 = new Thread(new TaskEvenOdd(print, 10, true));

t1.start();

t2.start();

}

}

class TaskEvenOdd implements Runnable {

private int max;

private Printer print;

private boolean isEvenNumber;

TaskEvenOdd(Printer print, int max, boolean isEvenNumber) {

this.print = print;

this.max = max;

this.isEvenNumber = isEvenNumber;

}

@Override

public void run() {

//System.out.println("Run method");

int number = isEvenNumber == true ? 2 : 1;

while (number <= max) {

if (isEvenNumber) {

//System.out.println("Even :"+ Thread.currentThread().getName());

// System.out.println( Thread.currentThread().getName());

print.printEven(number);

//number+=2;

} else {

// System.out.println( Thread.currentThread().getName());

print.printOdd(number);

// number+=2;

}

number += 2;

}

}

}

class Printer {

boolean isOdd = false;

synchronized void printEven(int number) {

while (isOdd == false) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.*out*.println(Thread.*currentThread*().getName()+" Even: " + number);

isOdd = false;

notifyAll();

}

synchronized void printOdd(int number) {

while (isOdd == true) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.*out*.println(Thread.*currentThread*().getName() +" Odd: " + number);

isOdd = true;

notifyAll();

}

}

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

package dayThree;

public class Q5WaitNotify {

public static void main(String[] args){

ThreadB b = new ThreadB();

b.start();

// System.out.println("I am main");

synchronized(b){

try{

System.*out*.println("Waiting for b to complete...");

b.wait();

}catch(InterruptedException e){

e.printStackTrace();

}

System.*out*.println("Total is: " + b.total);

}

System.*out*.println("I am main");

}

}

class ThreadB extends Thread{

int total;

@Override

public void run(){

synchronized(this){

for(int i=0; i<100 ; i++){

total += i;

}

notify();

}

}

}

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

package dayThree.Q6;

public class JoinSleep {

public static void main(String[] args) {

Thread t1 = new Thread(new MyRunnable(), "t1");

Thread t2 = new Thread(new MyRunnable(), "t2");

Thread t3 = new Thread(new MyRunnable(), "t3");

t1.start();

//start second thread after waiting for 2 seconds OR if it's dead

try {

t1.join(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

t2.start();

//start third thread only when second thread is dead

try {

// t1.join(12);

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

t3.start();

//let all threads finish execution before finishing main thread

try {

t1.join();

t2.join();

t3.join();

} catch (InterruptedException e) {

// *TODO Auto-generated catch block*

e.printStackTrace();

}

System.*out*.println("All threads are dead, exiting main thread");

}

}

class MyRunnable implements Runnable{

@Override

public void run() {

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

try {

//sleep duration for threads

Thread.*sleep*(4000);

// Thread.wait(4000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// System.out.println("thread running "+ Thread.currentThread().getName()+" "+Thread.currentThread().isAlive());

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

}

}

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

package dayThree;

import java.util.concurrent.\*;

import java.util.\*;

class CallableExample implements Callable

{

public Object call() throws Exception

{

Random generator = new Random();

Integer randomNumber = generator.nextInt(5);

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

return randomNumber;

}

}

public class CallableFutureTest

{

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

{

// FutureTask is a concrete class that

// implements both Runnable and Future

FutureTask[] randomNumberTasks = new FutureTask[5];

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

{

Callable callable = new CallableExample();

// Create the FutureTask with Callable

randomNumberTasks[i] = new FutureTask(callable);

// As it implements Runnable, create Thread

// with FutureTask

Thread t = new Thread(randomNumberTasks[i]);

t.start();

}

System.*out*.println("Printing using future get \n");

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

{

// As it implements Future, we can call get()

System.*out*.println(randomNumberTasks[i].get());

// This method blocks till the result is obtained

// The get method can throw checked exceptions

// like when it is interrupted. This is the reason

// for adding the throws clause to main

}

}

}

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

package dayThree;

import java.util.concurrent.Semaphore;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

class Connection {

private static Connection *instance* = new Connection();

private Semaphore sem = new Semaphore(5, true);

private int connections = 0;

private Connection() {}

public static Connection getInstance() {

return *instance*;

}

public void connect() {

try {

sem.acquire();

} catch (InterruptedException e1) {

e1.printStackTrace();

}

try {

doConnect();

} finally {

sem.release();

}

}

public void doConnect() {

synchronized (this) {

connections++;

System.*out*.println("Current connections: " + connections);

}

try {

Thread.*sleep*(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

synchronized (this) {

connections--; }

}

}

public class Semaphores {

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

ExecutorService executor = Executors.*newCachedThreadPool*();

for(int i=0; i < 20; i++) {

executor.submit(new Runnable() {

public void run() {

Connection.*getInstance*().connect();

}

});

}

// System.out.println("g");

executor.shutdown();

executor.awaitTermination(1, TimeUnit.*DAYS*);

System.*out*.println("Main command printed after semaphores");

}

}

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

package dayThree;

import java.util.Random;

import java.util.concurrent.CountDownLatch;

class CountDownLatchExample {

public static void main(String[] args) {

CountDownLatch latch = new CountDownLatch(3);

Random random = new Random();

WorkerThread wt1 = new WorkerThread(latch, random.nextInt(5000));

WorkerThread wt2 = new WorkerThread(latch, random.nextInt(5000));

WorkerThread wt3 = new WorkerThread(latch, random.nextInt(5000));

new Thread(wt1, "WT-11").start();

new Thread(wt2, "WT-22").start();

new Thread(wt3, "WT-33").start();

try {

latch.await();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println("Finally main is executed");

}

}

class WorkerThread implements Runnable {

private CountDownLatch latch;

private int delay;

public WorkerThread(CountDownLatch latch, int delay) {

this.latch = latch;

this.delay = delay;

}

@Override

public void run() {

String name = Thread.*currentThread*().getName();

int delay = this.delay;

System.*out*.println(name + " sleeping for " + delay + " milliseconds.");

try {

Thread.*sleep*(delay);

} catch (InterruptedException e) {

e.printStackTrace(); }

this.latch.countDown();

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

}

}

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

package dayThree;

public class Q10Deadlock {

// Creating Object Locks

static Object *ObjectLock1* = new Object();

static Object *ObjectLock2* = new Object();

private static class ThreadName1 extends Thread {

public void run() {

synchronized (*ObjectLock1*) {

System.*out*.println("Thread 1: Has ObjectLock1");

/\* Adding sleep() method so that

Thread 2 can lock ObjectLock2 \*/

try {

Thread.*sleep*(100);

}

catch (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println("Thread 1: Waiting for ObjectLock 2");

/\*Thread 1 has ObjectLock1

but waiting for ObjectLock2\*/

synchronized (*ObjectLock2*) {

System.*out*.println("Thread 1: No DeadLock");

}

}

}

}

private static class ThreadName2 extends Thread {

public void run() {

synchronized (*ObjectLock2*) {

System.*out*.println("Thread 2: Has ObjectLock2");

/\* Adding sleep() method so that

Thread 1 can lock ObjectLock1 \*/

try {

Thread.*sleep*(100);

}

catch (InterruptedException e) {

e.printStackTrace();

}

System.*out*.println("Thread 2: Waiting for ObjectLock 1");

/\*Thread 2 has ObjectLock2

but waiting for ObjectLock1\*/

synchronized (*ObjectLock1*) {

System.*out*.println("Thread 2: No DeadLock");

}

}

}

}

public static void main(String args[]) {

ThreadName1 thread1 = new ThreadName1();

ThreadName2 thread2 = new ThreadName2();

thread1.start();

thread2.start();

}

}