**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

Department of Computer Science & Engineering

**Subject Name: Java Programming**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024-2025**

**Part - 7**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **32.** | Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.  **PROGRAM CODE :**   1. **BY EXTENDING THREAD CLASS:**   import java.util.\*;  class prc32\_1 extends Thread  {  public void run()  {  System.out.println("Hello, World.(using Thread class)");  }    public static void main(String args[])  {  prc32\_1 t=new prc32\_1();  t.start();  }  }  **OUTPUT:**     1. **By using runnable interface:**   import java.util.\*;  class prc32\_2 implements Runnable  {  public void run()  {  System.out.println("Hello, World.(using runnable thread)");  }    public static void main(String[] args)  {  prc32\_2 m1 = new prc32\_2();  Thread t1=new Thread(m1);  t1.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this code, we can conclude that threads can be created by extending the in-built thread class as well as using the runnable interface. |
| **33**. | Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.  **PROGRAM CODE :**  import java.util.Scanner;  // Thread class for calculating sum of a portion of numbers  class SumThread extends Thread {  private int start;  private int end;  private int partialSum;  // Constructor to define range of numbers this thread will handle  public SumThread(int start, int end) {  this.start = start;  this.end = end;  }  @Override  public void run() {  partialSum = 0;  for (int i = start; i <= end; i++) {  partialSum += i;  }  }  // Method to return the partial sum calculated by this thread  public int getPartialSum() {  return partialSum;  }  }  public class prc33\_1 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);    // Input N and number of threads  System.out.print("Enter the value of N (sum numbers from 1 to N): ");  int N = scanner.nextInt();    System.out.print("Enter the number of threads: ");  int numThreads = scanner.nextInt();  // Create an array to hold threads  SumThread[] threads = new SumThread[numThreads];  // Calculate the range of numbers each thread should handle  int range = N / numThreads;  int start = 1;  // Create and start threads  for (int i = 0; i < numThreads; i++) {  int end = (i == numThreads - 1) ? N : (start + range - 1); // Last thread takes the remaining range  threads[i] = new SumThread(start, end);  threads[i].start();  start = end + 1;  }  // Wait for all threads to finish and collect results  int totalSum = 0;  for (int i = 0; i < numThreads; i++) {  try {  threads[i].join(); // Wait for the thread to finish  totalSum += threads[i].getPartialSum(); // Add each thread's partial sum to total sum  } catch (InterruptedException e) {  System.out.println("Thread interrupted: " + e.getMessage());  }  }  // Display the final result  System.out.println("The sum of numbers from 1 to " + N + " is: " + totalSum);  }  }    **OUTPUT:**    **CONCLUSION:**  By this code, we learn about separating tasks and dividing them among threads running simultaneously to make the process quicker and generate the sum accordingly. |
| **34.** | Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.  **PROGRAM CODE:**  import java.util.\*;  class GeneratorThread extends Thread  {  public void run()  {  for(int number=0;number<=10;number++)  {  System.out.println("Generated: " + number);  if (number % 2 == 0)  {  SquareThread.square(number);  }  else  {  CubeThread.cube(number);  }    try  {  Thread.sleep(1000);  }  catch (InterruptedException e)  {  e.printStackTrace();  }  }  }  }  class SquareThread extends Thread  {  public static void square(int number)  {  int square = number \* number;  System.out.println("Square: " + square);  }  }  class CubeThread extends Thread  {  public static void cube(int number)  {  int cube = number \* number \* number;  System.out.println("Cube: " + cube);  }  }  public class prc34\_1  {  public static void main(String[] args)  {  GeneratorThread generatorThread = new GeneratorThread();  generatorThread.start();  SquareThread squareThread = new SquareThread();  squareThread.start();  CubeThread cubeThread = new CubeThread();  cubeThread.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this problem, we learn that multiple threads can work together concurrently, generating numbers, checking if they are even or odd and give the output accordingly. |
| **35.** | Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.  **PROGRAM CODE:**  import java.util.\*;  class MyThread extends Thread  {  public void run()  {  for(int i=0;i<=10;i++)  {  System.out.println(i);  try  {  sleep(1000);  }  catch(InterruptedException e)  {  e.printStackTrace();  }  }  }  }  class prc35  {  public static void main(String args[])  {  MyThread t=new MyThread();  t.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this program, we learn to use the sleep method which delays the action of the code to manage the thread tasks as per our convenience. |
| **36.** | Write a program to create three threads ‘FIRST’, ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’ thread to 3, the ‘SECOND’ thread to 5(default) and the ‘THIRD’ thread to 7.  **PROGRAM CODE:**  import java.util.\*;  class first extends Thread  {  public void run()  {  System.out.println(first.class);  }  }  class second extends Thread  {  public void run()  {  System.out.println(second.class);  }  }  class third extends Thread  {  public void run()  {  System.out.println(third.class);  }  }  class prc36 extends Thread  {  public static void main(String args[])  {  first f1=new first();  second s1=new second();  third t1=new third();  f1.setPriority(3);  s1.setPriority(5);  t1.setPriority(7);  f1.start();  s1.start();  t1.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this program, we learn about setting the priorities of threads as per our requirements. |
| **37.** | Write a program to solve producer-consumer problem using thread synchronization.  **PROGRAM CODE:**  import java.util.LinkedList;  import java.util.Queue;  import java.util.Scanner;  // Shared buffer class with synchronization  class SharedBuffer {  private Queue<Integer> buffer = new LinkedList<>();  private int capacity;  public SharedBuffer(int capacity) {  this.capacity = capacity;  }  // Method for the producer to add items to the buffer  public synchronized void produce(int item) throws InterruptedException {  while (buffer.size() == capacity) {  wait(); // Wait if the buffer is full  }  buffer.add(item);  System.out.println("Produced: " + item);  notifyAll(); // Notify the consumer that an item has been produced  }  // Method for the consumer to take items from the buffer  public synchronized int consume() throws InterruptedException {  while (buffer.isEmpty()) {  wait(); // Wait if the buffer is empty  }  int item = buffer.poll();  System.out.println("Consumed: " + item);  notifyAll(); // Notify the producer that space is available in the buffer  return item;  }  }  // Producer thread class  class Producer extends Thread {  private SharedBuffer buffer;  private int itemsToProduce;  public Producer(SharedBuffer buffer, int itemsToProduce) {  this.buffer = buffer;  this.itemsToProduce = itemsToProduce;  }  @Override  public void run() {  try {  for (int i = 0; i < itemsToProduce; i++) {  buffer.produce(i);  Thread.sleep(500); // Simulate time taken to produce an item  }  } catch (InterruptedException e) {  System.out.println("Producer interrupted.");  }  }  }  // Consumer thread class  class Consumer extends Thread {  private SharedBuffer buffer;  private int itemsToConsume;  public Consumer(SharedBuffer buffer, int itemsToConsume) {  this.buffer = buffer;  this.itemsToConsume = itemsToConsume;  }  @Override  public void run() {  try {  for (int i = 0; i < itemsToConsume; i++) {  buffer.consume();  Thread.sleep(1000); // Simulate time taken to consume an item  }  } catch (InterruptedException e) {  System.out.println("Consumer interrupted.");  }  }  }  public class prac\_37 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  // Input for buffer capacity  System.out.print("Enter the buffer capacity: ");  int bufferCapacity = scanner.nextInt();  // Input for the number of items to produce and consume  System.out.print("Enter the number of items to produce: ");  int itemsToProduce = scanner.nextInt();  System.out.print("Enter the number of items to consume: ");  int itemsToConsume = scanner.nextInt();  // Create shared buffer  SharedBuffer sharedBuffer = new SharedBuffer(bufferCapacity);  // Create and start producer and consumer threads  Producer producer = new Producer(sharedBuffer, itemsToProduce);  Consumer consumer = new Consumer(sharedBuffer, itemsToConsume);  producer.start();  consumer.start();  try {  // Wait for both threads to complete execution  producer.join();  consumer.join();  } catch (InterruptedException e) {  System.out.println("Main thread interrupted.");  }  System.out.println("Producer and Consumer execution completed.");  }  }  **OUTPUT:**    **CONCLUSION:**  Here, we learn about using the synchronise keyword as well as join and notifyall methods. |