**PART-VII Multithreading**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| 32. | AIM :- Write a program to create thread which display “HelloWorld” message. A. by extending Thread class B. by usingRunnable interface. **PROGRAM CODE :**  class HelloWorldThread extends Thread {  public void run() {  System.out.println("Hello World");  }  }  class HelloWorldRunnable implements Runnable {  public void run() {  System.out.println("Hello World");  }  }  public class jprac\_32 {  public static void main(String[] args) {  // By Extending Thread Class  HelloWorldThread thread1 = new HelloWorldThread();  thread1.start();  // By Implementing Runnable Interface  Thread thread2 = new Thread(new HelloWorldRunnable());  thread2.start();  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates two ways to create and run threads in Java: by extending the Thread class and by implementing the Runnable interface. Both methods allow concurrent execution of code, but implementing Runnable is generally preferred as it allows the class to extend other classes and promotes better design by separating the thread's task from the thread's execution. |
| 33. | AIM : Write a program which takes N and number of threads asanargument. 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;  import java.util.Scanner;  class SumThread extends Thread {  private int start;  private int end;  private long partialSum;  public SumThread(int start, int end) {  this.start = start;  this.end = end;  this.partialSum = 0;  }  @Override  public void run() {  for (int i = start; i <= end; i++) {  partialSum += i;  }  }  public long getPartialSum() {  return partialSum;  }  }  public class jprac\_33 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the value of N: ");  int N = scanner.nextInt();  System.out.print("Enter the number of threads: ");  int numberOfThreads = scanner.nextInt();  if (N <= 0 || numberOfThreads <= 0) {  System.out.println("N and number\_of\_threads must be positive integers.");  scanner.close();  return;  }  SumThread[] threads = new SumThread[numberOfThreads];  int range = N / numberOfThreads;  int start = 1;  for (int i = 0; i < numberOfThreads; i++) {  int end = (i == numberOfThreads - 1) ? N : start + range - 1;  threads[i] = new SumThread(start, end);  threads[i].start();  start = end + 1;  }  long finalSum = 0;  try {  for (SumThread thread : threads) {  thread.join();  finalSum += thread.getPartialSum();  }  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("The sum of numbers from 1 to " + N + " is: " + finalSum);  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  scanner.close();  }  }  **OUTPUT**:    **CONCLUSION**:  This practical demonstrates how to use multithreading in Java to distribute the task of summing numbers from 1 to N across multiple threads. By dividing the range of numbers among the threads and collecting their partial sums, the program efficiently computes the final sum. This approach leverages the power of concurrent execution to handle large computations more effectively. |
| 34. | AIM : Write a java program that implements a multi-threadapplication that has three threads. First thread generatesrandom integer every 1 second and if the value is even,second thread computes the square of the number andprints. If the value is odd, the third thread will print thevalue of cube of the number.PROGRAM CODE : import java.util.Random;  class number extends Thread {  public int number;  public boolean isEven;  public void run() {  Random random = new Random();  while (true) {  number = random.nextInt(100);  isEven = (number % 2 == 0);  System.out.println("Generated Number: " + number);  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println(e);  }  }  }  }  class SquareCalculator extends Thread {  private number generator;  public SquareCalculator(number generator) {  this.generator = generator;  }  public void run() {  while (true) {  if (generator.isEven) {  int square = generator.number \* generator.number;  System.out.println("Square of " + generator.number + ": " + square);  }  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println(e);  }  }  }  }  class CubeCalculator extends Thread {  private number generator;  public CubeCalculator(number generator) {  this.generator = generator;  }  public void run() {  while (true) {  if (!generator.isEven) {  int cube = generator.number \* generator.number \* generator.number;  System.out.println("Cube of " + generator.number + ": " + cube);  }  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  System.out.println(e);  }  }  }  }  public class jprac\_34 {  public static void main(String[] args) {  number generator = new number();  SquareCalculator squareCalculator = new SquareCalculator(generator);  CubeCalculator cubeCalculator = new CubeCalculator(generator);  generator.start();  squareCalculator.start();  cubeCalculator.start();  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates the use of multithreading in Java to perform different tasks based on the generated random integer. The program uses three threads: one to generate random integers, one to compute the square of even numbers, and one to compute the cube of odd numbers. This approach showcases how threads can communicate and coordinate using shared data, enhancing the program's efficiency and responsiveness. |

**PART-VII Multithreading**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| 35. | AIM :- Write a program to increment the value of one variable byone and display it after one second using thread using sleep() method. **PROGRAM CODE :**  public class jprac\_35 extends Thread {  private int value;  public jprac\_35(int value) {  this.value = value;  }  public void run() {  try {  Thread.sleep(1000); // Sleep for one second  value++; // Increment the value  System.out.println("Value after increment: " + value);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  public static void main(String[] args) {  int initialValue = 10; // Initial value of the variable  jprac\_35 thread = new jprac\_35(initialValue);  thread.start();  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates how to use the Thread class and the sleep() method in Java to pause the execution of a thread for a specified period. The program increments a variable by one and displays its value after a one-second delay, showcasing basic thread operations and synchronization. |
| 36. | AIM : 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 theTHIRD’ thread to 7.PROGRAM CODE : class f1 extends Thread {  public void run() {  System.out.println("first : " + this.getPriority());  }  }  class f2 extends Thread {  public void run() {  System.out.println("second : " + this.getPriority());  }  }  class f3 extends Thread {  public void run() {  System.out.println("third : " + this.getPriority());  }  }  public class jprac\_36 {  public static void main(String[] args) {  // Create thread instances  f1 first = new f1();  f2 second = new f2();  f3 third = new f3();  // Set thread names  first.setName("FIRST");  second.setName("SECOND");  third.setName("THIRD");  // Set thread priorities  first.setPriority(3);  second.setPriority(Thread.NORM\_PRIORITY); // Default priority is 5  third.setPriority(7);  // Start the threads  first.start();  second.start();  third.start();  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  }  }  **OUTPUT**:    **CONCLUSION**:  This practical demonstrates how to create and manage thread priorities in Java. The program creates three threads (FIRST, SECOND, THIRD) and sets their priorities to 3, 5 (default), and 7, respectively. By setting thread priorities, we can influence the order in which threads are scheduled for execution, although the exact behavior may depend on the underlying operating system and JVM implementation. |
| 37. | AIM : Write a program to solve producer-consumer problem using thread synchronization.PROGRAM CODE : class sourses {  private int data;  private boolean available = false;  public void produce(int value) throws InterruptedException {  while (available) {  wait();  }  data = value;  available = true;  notifyAll();  }  public int consume() throws InterruptedException {  while (!available) {  wait();  }  available = false;  notifyAll();  return data;  }  }  class producer extends Thread {  private sourses sourses;  private int n;  public producer(sourses sourses, int n) {  this.sourses = sourses;  this.n = n;  }  @Override  public void run() {  try {  for (int i = 0; i < n; i++) {  sourses.produce(i);  System.out.println("Produced: " + i);  }  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  class consumer extends Thread {  private sourses sourses;  private int n;  public consumer(sourses sourses, int n) {  this.sourses = sourses;  this.n = n;  }  @Override  public void run() {  try {  for (int i = 0; i < n; i++) {  int value = sourses.consume();  System.out.println("Consumed: " + value);  }  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  public class jprac\_37 {  public static void main(String[] args) {  int n = 10; // Example value for n  sourses sourses = new sourses();  producer producer = new producer(sourses, n);  consumer consumer = new consumer(sourses, n);  producer.start();  consumer.start();  System.out.print("\nName : DHAVAL DESAI \nID : 23DCS020 ");  } }OUTPUT:   **CONCLUSION:**  This practical demonstrates the producer-consumer problem using thread synchronization in Java. The program uses a shared buffer with a fixed capacity, where the producer thread generates items and the consumer thread consumes them. Synchronization is achieved using the wait() and notifyAll() methods to ensure that the producer waits when the buffer is full and the consumer waits when the buffer is empty. This approach ensures proper coordination between threads, preventing race conditions and ensuring data consistency. |