##### 13. Write a program to illustrate **Multithreading** and **Multitasking**.

import java.util.Scanner;

class MyThread **extends** Thread

{

private SharedResource resource;

private int numIncrements;

// Constructor to initialize shared resource and number of increments

public **MyThread**(SharedResource resource, int numIncrements)

{

this.resource = resource;

this.numIncrements = numIncrements;

}

@Override

public void **run**()

{

// Perform the specified number of increments

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

{

resource.increment();

}

}

// Driver code is given for your reference

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

{

Scanner scanner = new Scanner(System.in);

int numThreads = 0;

int numIncrements = 0;

boolean validInput = false;

while (!validInput)

{

try

{

System.out.println("Enter the number of threads:");

numThreads = Integer.parseInt(scanner.nextLine());

System.out.println("Enter the number of increments per thread:");

numIncrements = Integer.parseInt(scanner.nextLine());

if (numThreads >= 1 && numIncrements >= 1)

{

validInput = true;

}

else

{

System.out.println("Please enter valid numbers (1 or more) for threads and increments.");

}

}

catch (NumberFormatException e)

{

System.out.println("Please enter valid numbers.");

}

}

SharedResource resource = new SharedResource();

MyThread[] threads = new MyThread[numThreads];

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

{

threads[i] = new MyThread(resource, numIncrements);

threads[i].start();

}

scanner.close();

}

}

class **SharedResource**

{

private int count = 0;

public void increment()

{

synchronized (this)

{

count++;

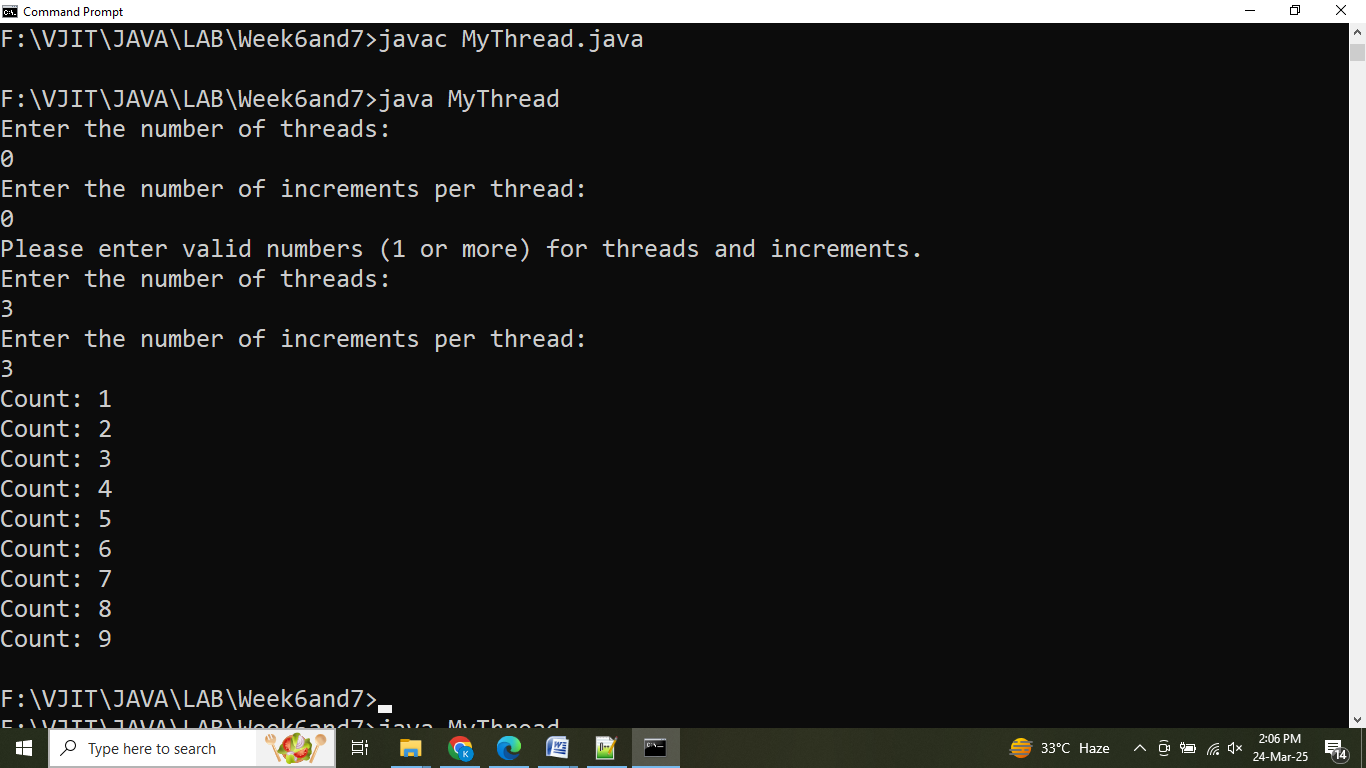
System.out.println("Count: " + count);

}

}

}

**Output:**



**14. Write a program to illustrate thread priorities.**

import java.util.Scanner;

class PriorityDemo **extends** Thread

{

@Override

public void **run**()

{

// Print thread name and its priority

System.out.println("Thread name: " + Thread.currentThread().getName() + ", Priority: " + Thread.currentThread().getPriority());

}

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

{

Scanner scanner = new Scanner(System.in);

PriorityDemo thread1 = new PriorityDemo();

PriorityDemo thread2 = new PriorityDemo();

System.out.println("Enter priority for Thread 1 (minimum/maximum):");

String priorityInput1 = scanner.nextLine().toLowerCase();

System.out.println("Enter priority for Thread 2 (minimum/maximum):");

String priorityInput2 = scanner.nextLine().toLowerCase();

int priority1 = getPriorityValue(priorityInput1);

int priority2 = getPriorityValue(priorityInput2);

thread1.setPriority(priority1);

thread2.setPriority(priority2);

thread1.start();

thread2.start();

scanner.close();

}

private static int **getPriorityValue**(String input)

{

if (input.equals("minimum"))

{

return Thread.MIN\_PRIORITY;

}

else if (input.equals("maximum"))

{

return Thread.MAX\_PRIORITY;

}

else

{

System.out.println("Invalid input. Setting priority to default.");

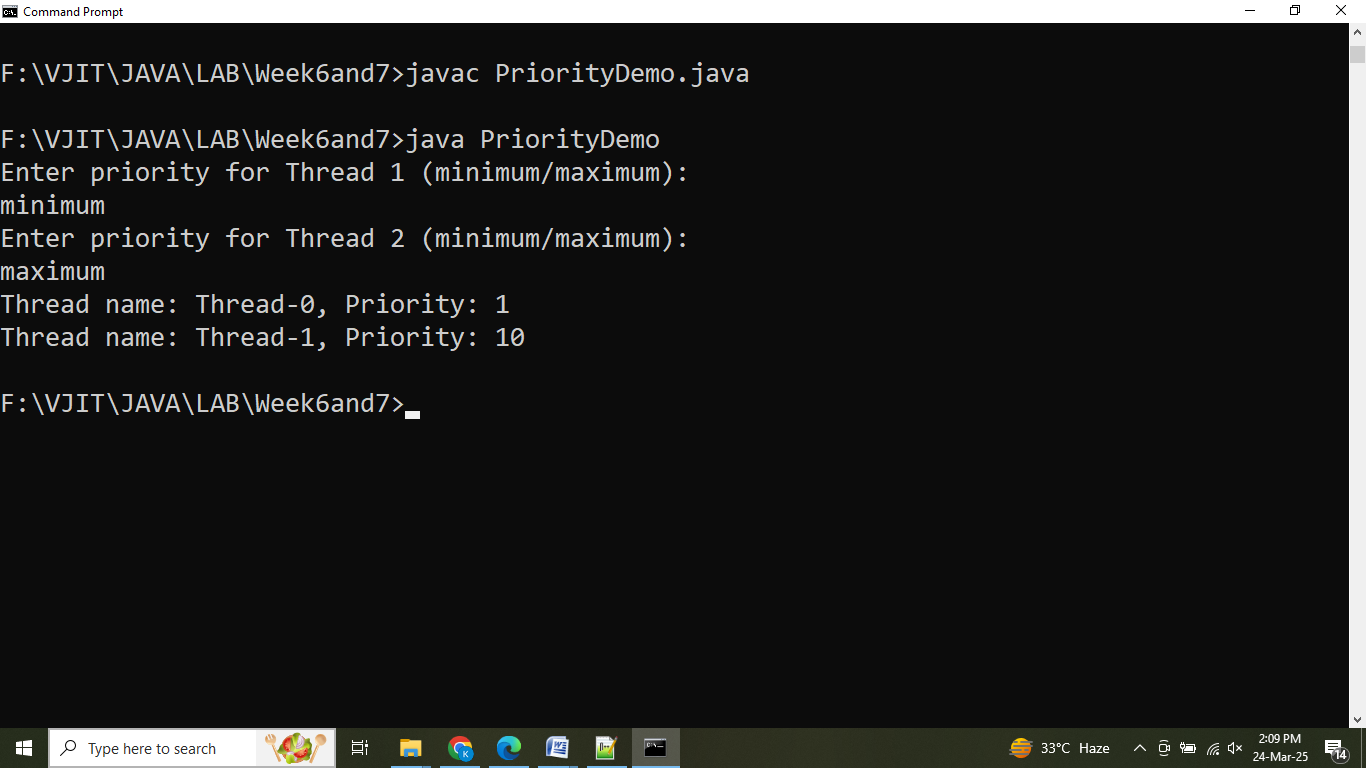
return Thread.NORM\_PRIORITY;

}

}

}

**Output:**



##### **15. 1 Write a program to illustrate Synchronization**

Write a Java program using synchronized threads, which demonstrates producer consumer concept.

The **Producer-Consumer** problem is a classic synchronization problem in computer science. It involves **two types** of threads:

* **Producer**: This thread generates data (e.g., items) and places it in a shared resource (like a buffer).
* **Consumer**: This thread takes data (items) from the shared resource for processing or consumption.

*The problem revolves around coordinating these threads such that:*

* The producer does not produce when the buffer is full.
* The consumer does not consume when the buffer is empty.
* Both threads operate safely without corrupting the shared resource.

To achieve synchronization in Java, we can use the synchronized keyword along with wait and notify methods to control the interaction between the threads.

import java.util.LinkedList;

import java.util.Scanner;

class **SharedBuffer**

{

private LinkedList<Integer> buffer = new LinkedList<>();

private int capacity;

// Constructor to set the buffer capacity

public **SharedBuffer**(int capacity)

{

this.capacity = capacity;

}

// Produce an item and add it 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);

notify(); // Notify the consumer that an item is available

}

// Consume an item from the buffer

public synchronized void **consume**() throws InterruptedException

{

while (buffer.isEmpty())

{

wait(); // Wait if the buffer is empty

}

int item = buffer.removeFirst();

System.out.println("Consumed: " + item);

notify(); // Notify the producer that space is available

}

}

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 = 1; i <= itemsToProduce; i++)

{

buffer.produce(i);

Thread.sleep(100); // Simulating some delay in producing

}

}

catch (InterruptedException e)

{

Thread.currentThread().interrupt();

}

}

}

class Consumer **extends** Thread

{

private SharedBuffer buffer;

public **Consumer**(SharedBuffer buffer)

{

this.buffer = buffer;

}

@Override

public void **run**()

{

try

{

// Continue consuming until the producer has finished producing

while (true)

{

buffer.consume();

Thread.sleep(100); // Simulating some delay in consuming

}

}

catch (InterruptedException e)

{

Thread.currentThread().interrupt();

}

}

}

public class **ProducerConsumerDemo**

{

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

{

Scanner scanner = new Scanner(System.in);

// Get user input for the number of items to produce

System.out.print("Enter the number of items to produce: ");

int itemsToProduce = scanner.nextInt();

// Create a shared buffer

SharedBuffer buffer = new SharedBuffer(itemsToProduce);

// Create producer and consumer threads

Producer producer = new Producer(buffer, itemsToProduce);

Consumer consumer = new Consumer(buffer);

// Start the threads

producer.start();

consumer.start();

// Wait for producer to finish producing before terminating the consumer

try

{

producer.join(); // Wait for producer to finish

}

catch (InterruptedException e)

{

Thread.currentThread().interrupt();

}

// Interrupt the consumer to gracefully stop the thread after production

consumer.interrupt();

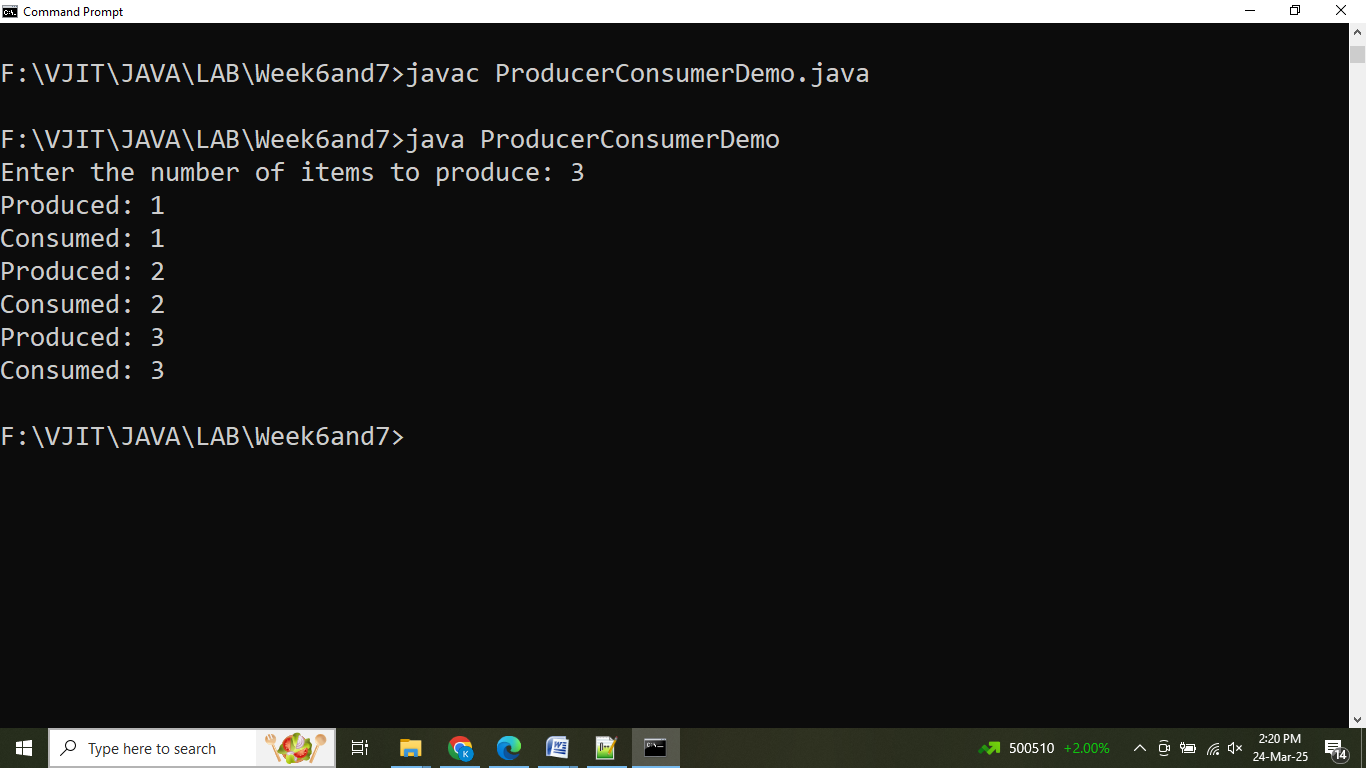
// Close the scanner

scanner.close();

}

}

**Output:**



**15.2** Printing tables using **synchronization** and threads

Create a Java program that utilizes multi-threading to generate multiplication tables.

import java.util.Scanner;

class TablePrinter **implements** Runnable

{

private int tableNumber;

public **TablePrinter**(int tableNumber)

{

this.tableNumber = tableNumber;

}

// Implement the run method to generate the multiplication table

@Override

public void **run**()

{

try

{

// Generate multiplication table for the assigned tableNumber

for (int i = 1; i <= 10; i++)

{

System.out.println(tableNumber + " \* " + i + " = " + (tableNumber \* i));

// Adding a small delay (100ms) for better visualization

Thread.sleep(100);

}

}

catch (InterruptedException e)

{

System.err.println("Thread was interrupted: " + e.getMessage());

}

}

}

public class **Main**

{

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

{

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of tables:");

int numTables = scanner.nextInt();

Thread[] threads = new Thread[numTables];

// Create and start threads for each table

for (int i = 1; i <= numTables; i++)

{

// Create a TablePrinter for each table (1 to numTables)

TablePrinter tablePrinter = new TablePrinter(i);

// Create a new thread for each TablePrinter instance

threads[i - 1] = new Thread(tablePrinter);

threads[i - 1].start(); // Start the thread

}

// Wait for all threads to finish using join()

try

{

for (Thread thread : threads)

{

// Ensure the main thread waits for each worker thread to complete

thread.join();

}

}

catch (InterruptedException e)

{

System.err.println("Main thread was interrupted: " + e.getMessage());

}

// Close the scanner

scanner.close();

}

}

**Output:**

