Ex no: 6	**Implement Programs Using Jdbc To Establish And Manage Database Connections For Data Persistence And Retrieval.**
Date:	

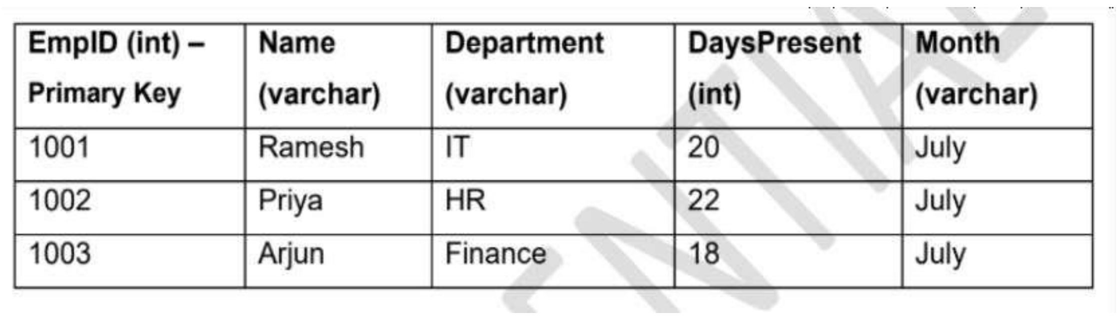
AIM:

To Implement programs using JDBC to establish and manage database connections for data persistence and retrieval.

QUESTION 1:

Employee Attendance Management

Write a Java program using JDBC to create the following employee attendance table:



EmpID (int) – Primary Key	Name (varchar)	Department (varchar)	DaysPresent (int)	Month (varchar)
1001	Ramesh	IT	20	July
1002	Priya	HR	22	July
1003	Arjun	Finance	18	July

Perform the following operations using JDBC Statement:

- Display all employees who attended more than 20 days in a month.
- Update the attendance of EmpID 1003 to 21.
- Display the average attendance of employees department-wise.
- Insert a new employee record and display the updated table.
- Delete all records of employees with attendance below 10 days.
- Display the employee(s) with the highest attendance in each department.

CODE:

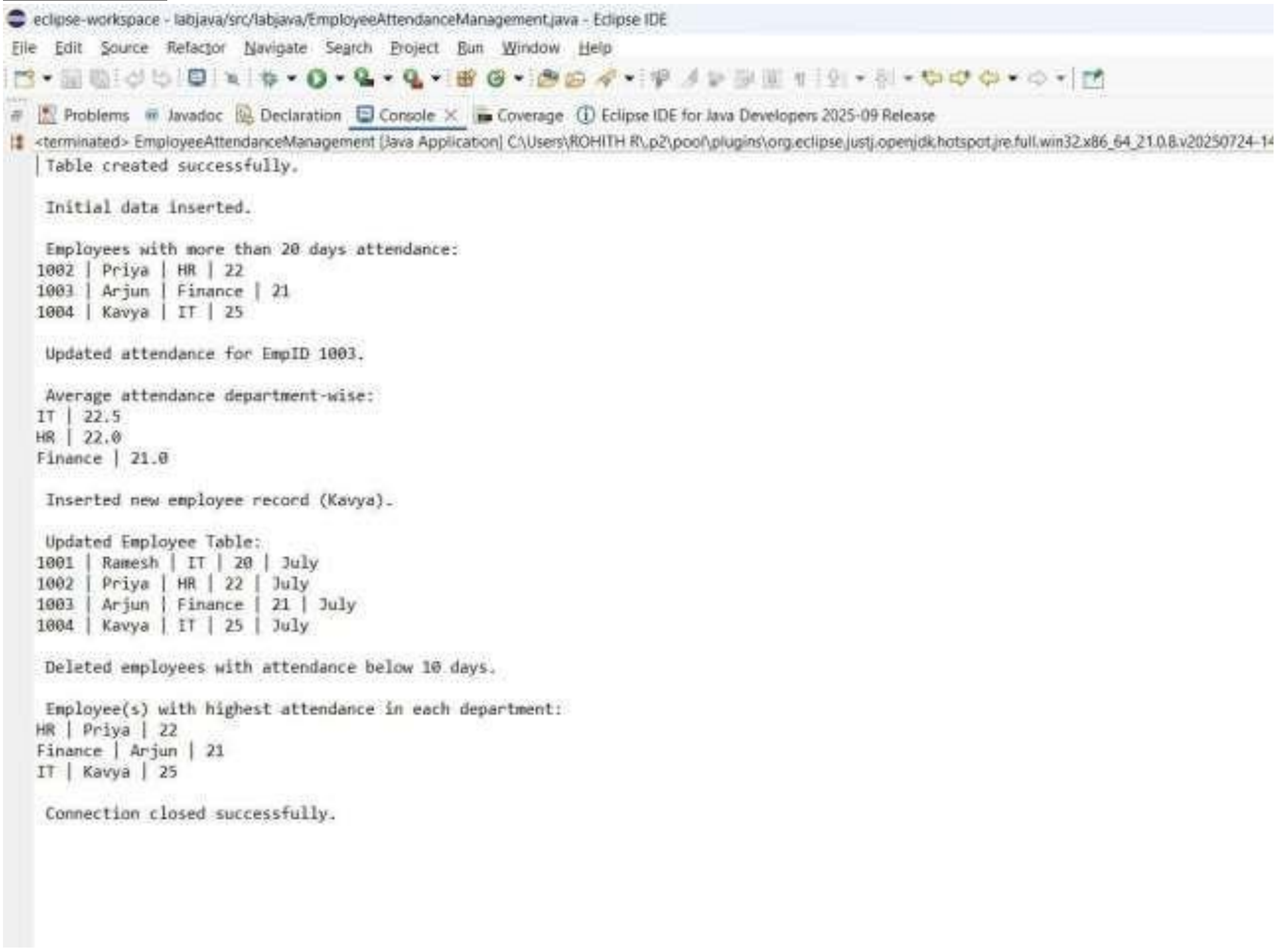
```
package labjava;

import java.sql.*;

public class EmployeeAttendanceManagement {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/company_db";
        String user = "root";
        String password = "password";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(url, user, password);
            Statement stmt = conn.createStatement();

            String createTable = """
                CREATE TABLE IF NOT EXISTS employee_attendance (
                    EmpID INT PRIMARY KEY,
                    Name VARCHAR(50),
                    Department VARCHAR(50),
                    DaysPresent INT,
                    Month VARCHAR(20)
                );
            """;
            stmt.executeUpdate(createTable);
            System.out.println("Table created successfully.\n");

            String insertRecords = """
                INSERT INTO employee_attendance (EmpID, Name, Department, DaysPresent, Month)
VALUES
                (1001, 'Ramesh', 'IT', 20, 'July'),
                (1002, 'Priya', 'HR', 22, 'July'),
                (1003, 'Arjun', 'Finance', 18, 'July')
                ON DUPLICATE KEY UPDATE Name=VALUES(Name);
            """;
            stmt.executeUpdate(insertRecords);
            System.out.println("Initial data inserted.\n");

            System.out.println("Employees with more than 20 days attendance:");
            ResultSet rs = stmt.executeQuery("SELECT * FROM employee_attendance WHERE
DaysPresent > 20;");
            while (rs.next()) {
                System.out.println(
                    rs.getInt("EmpID") + " | " +
                    rs.getString("Name") + " | " +
                    rs.getString("Department") + " | " +
                    rs.getInt("DaysPresent")
                );
            }
            System.out.println();

            stmt.executeUpdate("UPDATE employee_attendance SET DaysPresent = 21 WHERE
EmpID = 1003;");
            System.out.println("Updated attendance for EmpID 1003.\n");

            System.out.println("Average attendance department-wise:");
            rs = stmt.executeQuery(
                "SELECT Department, AVG(DaysPresent) AS AvgAttendance FROM
employee_attendance GROUP BY Department;"
            );
            while (rs.next()) {
                System.out.println(rs.getString("Department") + " | " + rs.getDouble("AvgAttendance"));
            }
```

```
            System.out.println();

            stmt.executeUpdate("DELETE FROM employee_attendance WHERE EmpID = 1004;");
            stmt.executeUpdate("INSERT INTO employee_attendance VALUES (1004, 'Kavya', 'IT', 25,
'July');");
            System.out.println("Inserted new employee record (Kavya).");

            rs = stmt.executeQuery("SELECT * FROM employee_attendance;");
            System.out.println("\nUpdated Employee Table:");
            while (rs.next()) {
                System.out.println(
                    rs.getInt("EmpID") + " | " +
                    rs.getString("Name") + " | " +
                    rs.getString("Department") + " | " +
                    rs.getInt("DaysPresent") + " | " +
                    rs.getString("Month")
                );
            }
            System.out.println();

            stmt.executeUpdate("DELETE FROM employee_attendance WHERE DaysPresent < 10;");
            System.out.println("Deleted employees with attendance below 10 days.\n");

            System.out.println("Employee(s) with highest attendance in each department:");
            String highestQuery = """
                SELECT e.Department, e.Name, e.DaysPresent
                FROM employee_attendance e
                WHERE e.DaysPresent = (
                    SELECT MAX(DaysPresent)
                    FROM employee_attendance
                    WHERE Department = e.Department
                );
            """;
            rs = stmt.executeQuery(highestQuery);
            while (rs.next()) {
                System.out.println(
                    rs.getString("Department") + " | " +
                    rs.getString("Name") + " | " +
                    rs.getInt("DaysPresent")
                );
            }

            conn.close();
            System.out.println("\nConnection closed successfully.");

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

SQL QUERY:

CREATE DATABASE company_db;
USE company_db;
SHOW DATABASES;

OUTPUT:


eclipse-workspace - labjava/src/labjava/EmployeeAttendanceManagement.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> EmployeeAttendanceManagement [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20250724-14
Table created successfully.

 Initial data inserted.

 Employees with more than 20 days attendance:
1002 | Priya | HR | 22
1003 | Arjun | Finance | 21
1004 | Kavya | IT | 25

 Updated attendance for EmpID 1003.

 Average attendance department-wise:
IT | 22.5
HR | 22.0
Finance | 21.0

 Inserted new employee record (Kavya).

 Updated Employee Table:
1001 | Ramesh | IT | 20 | July
1002 | Priya | HR | 22 | July
1003 | Arjun | Finance | 21 | July
1004 | Kavya | IT | 25 | July

 Deleted employees with attendance below 10 days.

 Employee(s) with highest attendance in each department:
HR | Priya | 22
Finance | Arjun | 21
IT | Kavya | 25

 Connection closed successfully.
```

QUESTION 2:

Inventory Management System

Write a Java program using JDBC to create the following inventory table for an online store:

ProductID (int) Primary Key	ProductName (varchar)	Category (varchar)	Stock (int)	Price (int)
2001	Laptop	Electronics	15	55000
2002	Chair	Furniture	40	3000
2003	Headphones	Electronics	25	2500

Perform the following operations using JDBC PreparedStatement:

- Insert a new product into the table.
- Update the price of a given product.
- Delete a product by ProductID.
- Display all products with stock less than 20.
- Display the total stock value of all products (stock × price).
- Display the most expensive and cheapest product.

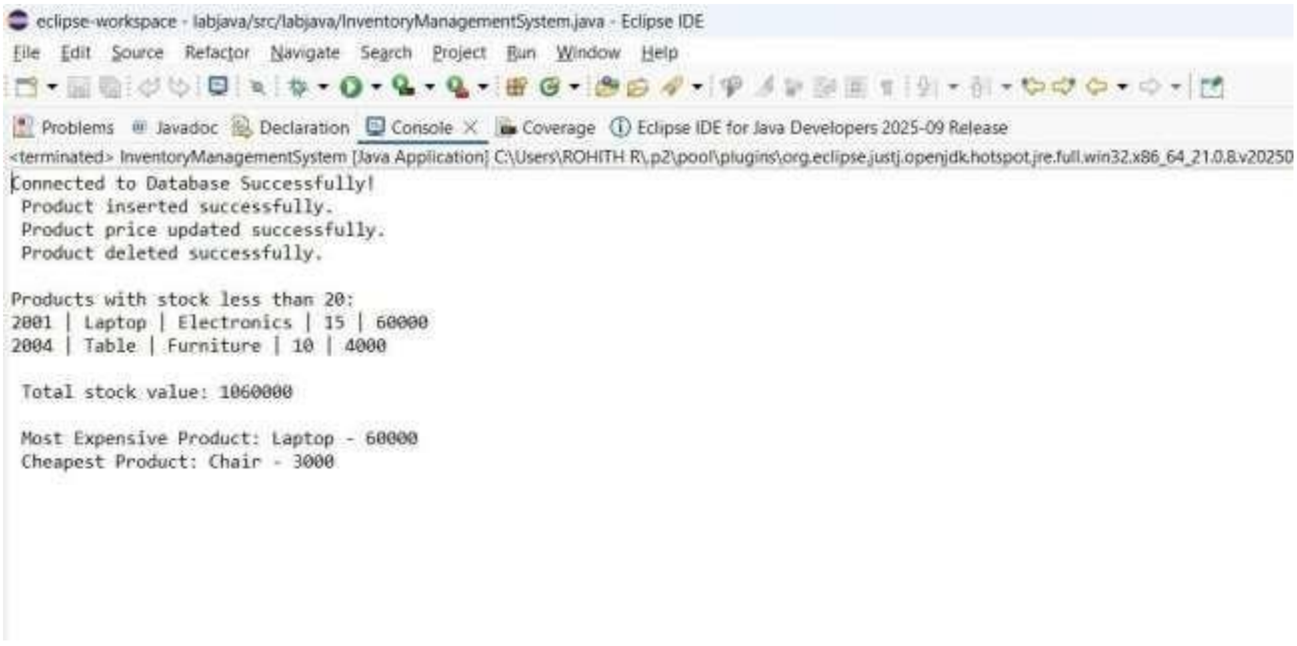
CODE:

```
 package labjava;

import java.sql.*;
import java.util.*;

public class InventoryManagementSystem {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/inventory_db";
String user = "root";
String password = "password";

try (Connection con = DriverManager.getConnection(url, user, password);
Scanner sc = new Scanner(System.in)) {

System.out.println("Connected to Database Successfully!");

String insertSQL = "INSERT INTO inventory VALUES (?, ?, ?, ?, ?)";
PreparedStatement insertStmt = con.prepareStatement(insertSQL);
insertStmt.setInt(1, 2004);
insertStmt.setString(2, "Table");
insertStmt.setString(3, "Furniture");
insertStmt.setInt(4, 10);
insertStmt.setInt(5, 4000);
insertStmt.executeUpdate();
System.out.println(" Product inserted successfully.");

String updateSQL = "UPDATE inventory SET Price=? WHERE ProductID=?";
PreparedStatement updateStmt = con.prepareStatement(updateSQL);
updateStmt.setInt(1, 60000);
updateStmt.setInt(2, 2001);
updateStmt.executeUpdate();
System.out.println(" Product price updated successfully.");

String deleteSQL = "DELETE FROM inventory WHERE ProductID=?";
PreparedStatement deleteStmt = con.prepareStatement(deleteSQL);
deleteStmt.setInt(1, 2003);
deleteStmt.executeUpdate();
System.out.println(" Product deleted successfully.");
System.out.println("\nProducts with stock less than 20:");
String selectLowStockSQL = "SELECT * FROM inventory WHERE Stock < 20";
PreparedStatement lowStockStmt = con.prepareStatement(selectLowStockSQL);
ResultSet rs1 = lowStockStmt.executeQuery();
while (rs1.next()) {
System.out.println(rs1.getInt("ProductID") + " | " +
rs1.getString("ProductName") + " | " +
rs1.getString("Category") + " | " +
rs1.getInt("Stock") + " | " +
rs1.getInt("Price"));
}

String totalValueSQL = "SELECT SUM(Stock * Price) AS TotalValue FROM inventory";
PreparedStatement totalStmt = con.prepareStatement(totalValueSQL);
ResultSet rs2 = totalStmt.executeQuery();
if (rs2.next()) {
System.out.println("\n Total stock value: " + rs2.getInt("TotalValue"));
}

String expensiveSQL = "SELECT * FROM inventory ORDER BY Price DESC LIMIT 1";
String cheapSQL = "SELECT * FROM inventory ORDER BY Price ASC LIMIT 1";

ResultSet rs3 = con.prepareStatement(expensiveSQL).executeQuery();
ResultSet rs4 = con.prepareStatement(cheapSQL).executeQuery();

if (rs3.next()) {
System.out.println("\n Most Expensive Product: " + rs3.getString("ProductName") + " - " +
rs3.getInt("Price"));
}
if (rs4.next()) {
System.out.println(" Cheapest Product: " + rs4.getString("ProductName") + " - " +
rs4.getInt("Price"));
}

} catch (SQLException e) {
e.printStackTrace();
}
}
}
```

SQL QUERY:

```
CREATE DATABASE inventory_db;
USE inventory_db;


CREATE TABLE inventory (
    ProductID INT PRIMARY KEY,
    ProductName VARCHAR(50),
    Category VARCHAR(50),
    Stock INT,
    Price INT
);

INSERT INTO inventory VALUES
(2001, 'Laptop', 'Electronics', 15, 55000),
(2002, 'Chair', 'Furniture', 40, 3000),
(2003, 'Headphones', 'Electronics', 25, 2500);
DELETE FROM inventory WHERE ProductID = 2004;
```

OUTPUT:

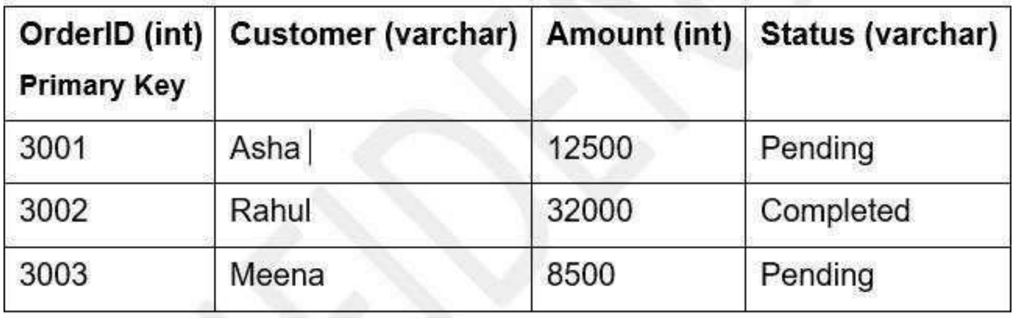

eclipse-workspace - labjava/src/labjava/InventoryManagementSystem.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> InventoryManagementSystem [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20250
Connected to Database Successfully!
 Product inserted successfully.
 Product price updated successfully.
 Product deleted successfully.

Products with stock less than 20:
2001 | Laptop | Electronics | 15 | 60000
2004 | Table | Furniture | 10 | 4000

 Total stock value: 1060000

 Most Expensive Product: Laptop - 60000
 Cheapest Product: Chair - 3000
```

QUESTION 3:

Customer Orders Tracking

Write a Java program using **JDBC CallableStatement** to manage the following customer orders table:



OrderID (int) Primary Key	Customer (varchar)	Amount (int)	Status (varchar)
3001	Asha	12500	Pending
3002	Rahul	32000	Completed
3003	Meena	8500	Pending

Create a Stored Procedure in the database to perform operations. Call the procedure using CallableStatement in Java.

- Tasks for the stored procedure:
- Display all pending orders.
- Update the status of a given order to “Completed”.
- Display total sales amount.
- Display all customers who placed orders greater than Rs. 20,000.
- Display the customer with the largest order.

CODE:

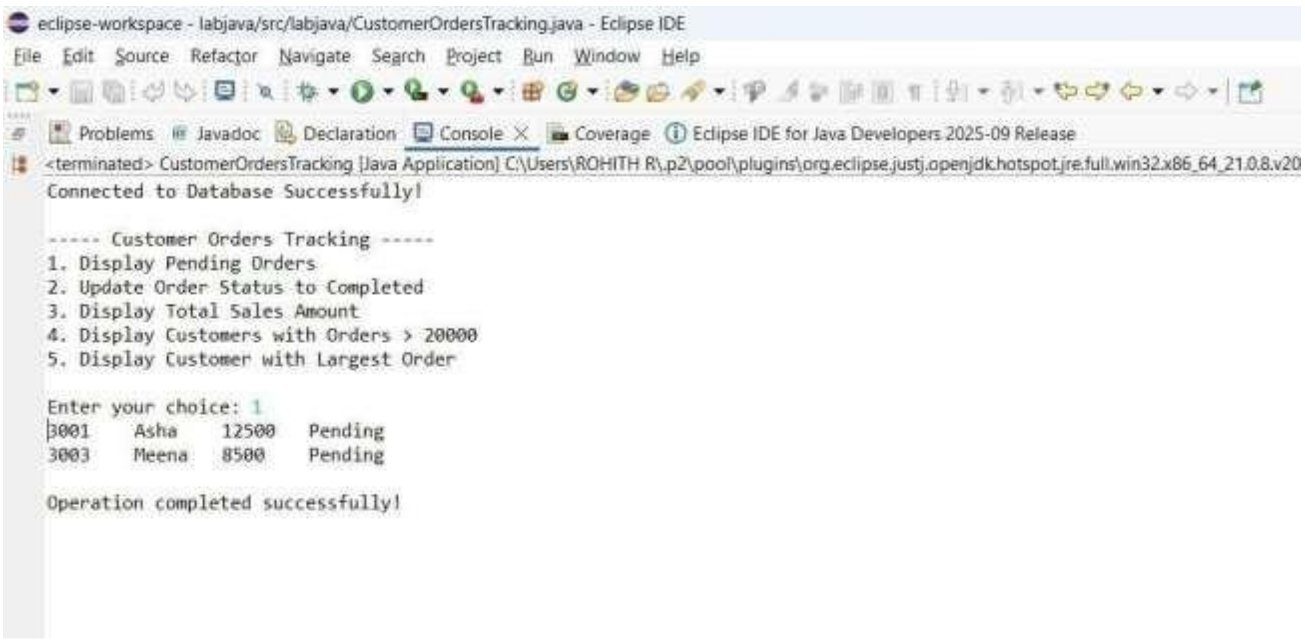
```
package labjava;

import java.sql.*;
import java.util.Scanner;

public class CustomerOrdersTracking {
public static void main(String[] args) {
String url = "jdbc:mysql://localhost:3306/orders_db";
String user = "root";
String password = "password";

try (Connection conn = DriverManager.getConnection(url, user, password);
Scanner sc = new Scanner(System.in)) {

Class.forName("com.mysql.cj.jdbc.Driver");
System.out.println("Connected to Database Successfully!\n");

System.out.println("----- Customer Orders Tracking------ ");
System.out.println("1. Display Pending Orders");
System.out.println("2. Update Order Status to Completed");
System.out.println("3. Display Total Sales Amount");
System.out.println("4. Display Customers with Orders > 20000");
System.out.println("5. Display Customer with Largest Order");
System.out.print("\nEnter your choice: ");
int choice = sc.nextInt();

int orderId = 0;
if (choice == 2) {
System.out.print("Enter OrderID to update: ");
orderId = sc.nextInt();
}

CallableStatement cs = conn.prepareCall("{call manage_orders(?, ?)}");
cs.setInt(1, choice);
cs.setInt(2, orderId);

boolean hasResult = cs.execute();
while (hasResult) {
ResultSet rs = cs.getResultSet();
ResultSetMetaData meta = rs.getMetaData();
int cols = meta.getColumnCount();



while (rs.next()) {
for (int i = 1; i <= cols; i++) {
System.out.print(rs.getString(i) + "\t");
}
System.out.println();
}
hasResult = cs.getMoreResults();
}

System.out.println("\nOperation completed successfully!");
conn.close();

} catch (Exception e) {
e.printStackTrace();
}
}
}
```

SQL QUERY:

```
  CREATE DATABASE IF NOT EXISTS orders_db; USE orders_db;

CREATE TABLE IF NOT EXISTS customer_orders ( OrderID INT PRIMARY KEY, Customer
VARCHAR(50), Amount INT, Status VARCHAR(20) );

INSERT INTO customer_orders VALUES (3001, 'Asha', 12500, 'Pending'), (3002, 'Rahul', 32000,
'Completed'), (3003, 'Meena', 8500, 'Pending');

DELIMITER $$

CREATE PROCEDURE manage_orders(IN choice INT, IN givenOrderID INT) BEGINIF choice =
1 THEN
SELECT * FROM customer_orders WHERE Status = 'Pending';


ELSEIF choice = 2 THEN
   UPDATE customer_orders SET Status = 'Completed' WHERE OrderID = givenOrderID;
   SELECT CONCAT('Order ', givenOrderID, ' marked as Completed.') AS Message;


ELSEIF choice = 3 THEN
   SELECT SUM(Amount) AS Total_Sales FROM customer_orders;
```

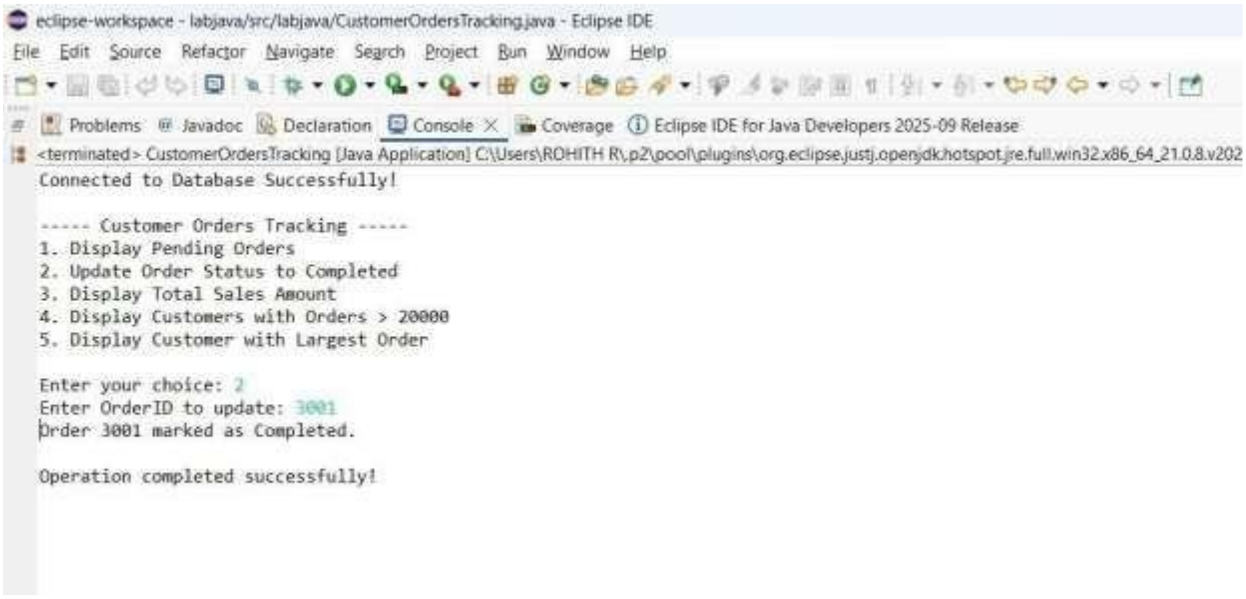
```
ELSEIF choice = 4 THEN
    SELECT Customer, Amount FROM customer_orders WHERE Amount > 20000;


ELSEIF choice = 5 THEN
    SELECT Customer, Amount FROM customer_orders
    WHERE Amount = (SELECT MAX(Amount) FROM customer_orders);
END IF;

END$$

DELIMITER ;
```

OUTPUT:



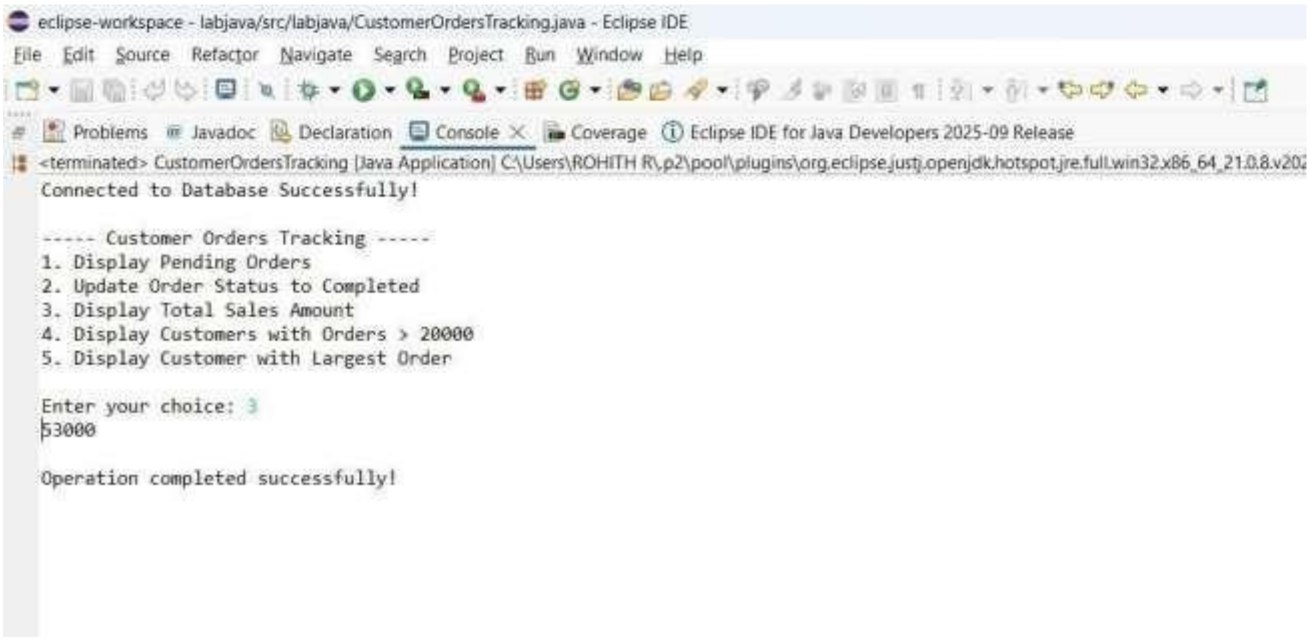
eclipse-workspace - labjava/src/labjava/CustomerOrdersTracking.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> CustomerOrdersTracking [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20
Connected to Database Successfully!

----- Customer Orders Tracking -----
1. Display Pending Orders
2. Update Order Status to Completed
3. Display Total Sales Amount
4. Display Customers with Orders > 20000
5. Display Customer with Largest Order

Enter your choice: 1
3001    Asha    12500   Pending
3003    Meena   8500    Pending

Operation completed successfully!
```

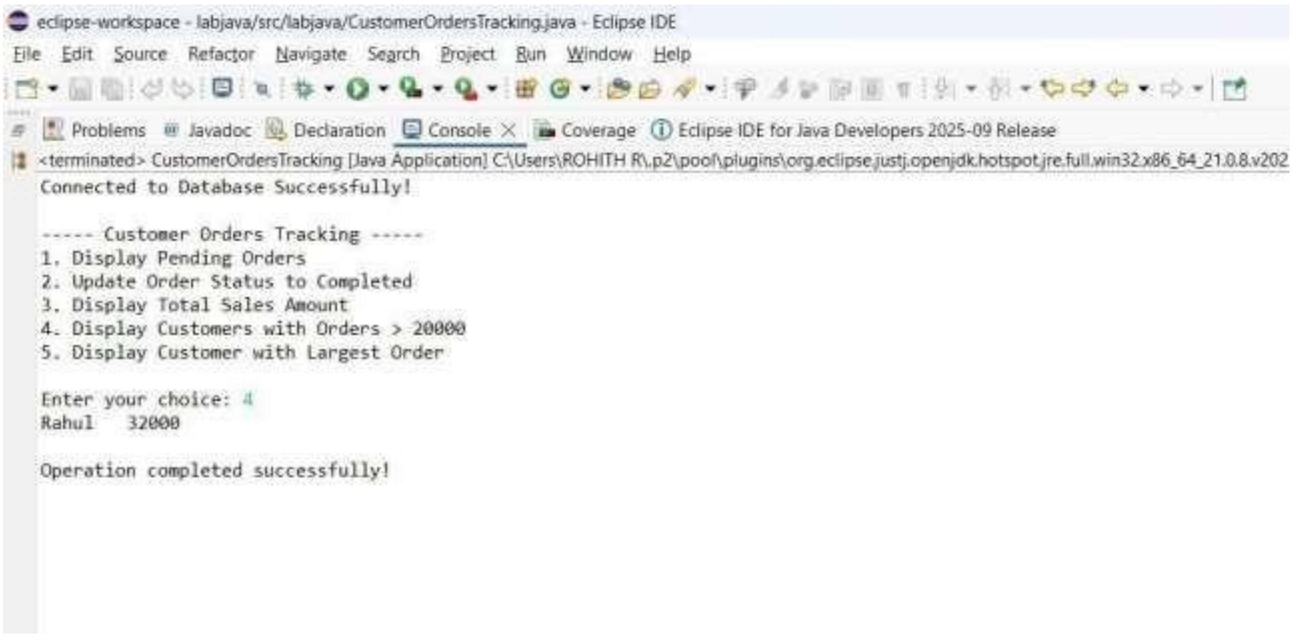


eclipse-workspace - labjava/src/labjava/CustomerOrdersTracking.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> CustomerOrdersTracking [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v202
Connected to Database Successfully!

----- Customer Orders Tracking -----
1. Display Pending Orders
2. Update Order Status to Completed
3. Display Total Sales Amount
4. Display Customers with Orders > 20000
5. Display Customer with Largest Order

Enter your choice: 2
Enter OrderID to update: 3001
Order 3001 marked as Completed.

Operation completed successfully!
```


eclipse-workspace - labjava/src/labjava/CustomerOrdersTracking.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> CustomerOrdersTracking [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v202
Connected to Database Successfully!

----- Customer Orders Tracking -----
1. Display Pending Orders
2. Update Order Status to Completed
3. Display Total Sales Amount
4. Display Customers with Orders > 20000
5. Display Customer with Largest Order

Enter your choice: 3
53000

Operation completed successfully!
```


eclipse-workspace - labjava/src/labjava/CustomerOrdersTracking.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> CustomerOrdersTracking [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v202
Connected to Database Successfully!

----- Customer Orders Tracking -----
1. Display Pending Orders
2. Update Order Status to Completed
3. Display Total Sales Amount
4. Display Customers with Orders > 20000
5. Display Customer with Largest Order

Enter your choice: 4
Rahul	32000

Operation completed successfully!
```

RESULT:

Thus the java program of using JDBC to establish and manage database connections for data persistence and retrieval was successfully executed and output was verified.

Ex no: 7	**Develop programs using multithreading to achieve concurrent execution and improve application performance**
Date:	

AIM:

To Develop programs using multithreading to achieve concurrent execution and improve application performance.

QUESTION 1:

Write a Java program that reads one 5-digit number and one sentence, then runs six threads at the same time.

Each thread must print START and FINISH with its thread name. After all threads complete, print “Done”.

Subtasks

- Create a thread to print the sum of the digits of the 5-digit number.
- Create a thread to print “hello world!” five times, pausing briefly between prints.
- Create a thread to print the sentence without vowels (a, e, i, o, u; ignore case).
- Create a thread to print the count of each character in the sentence (ignore spaces; treat uppercase/lowercase as the same).
- Create a thread to print numbers 1 2 3 4 5 (ascending).
- Create a thread to print numbers 5 4 3 2 1 (descending).
  Hint:
- Start all threads together; outputs may appear in mixed order.

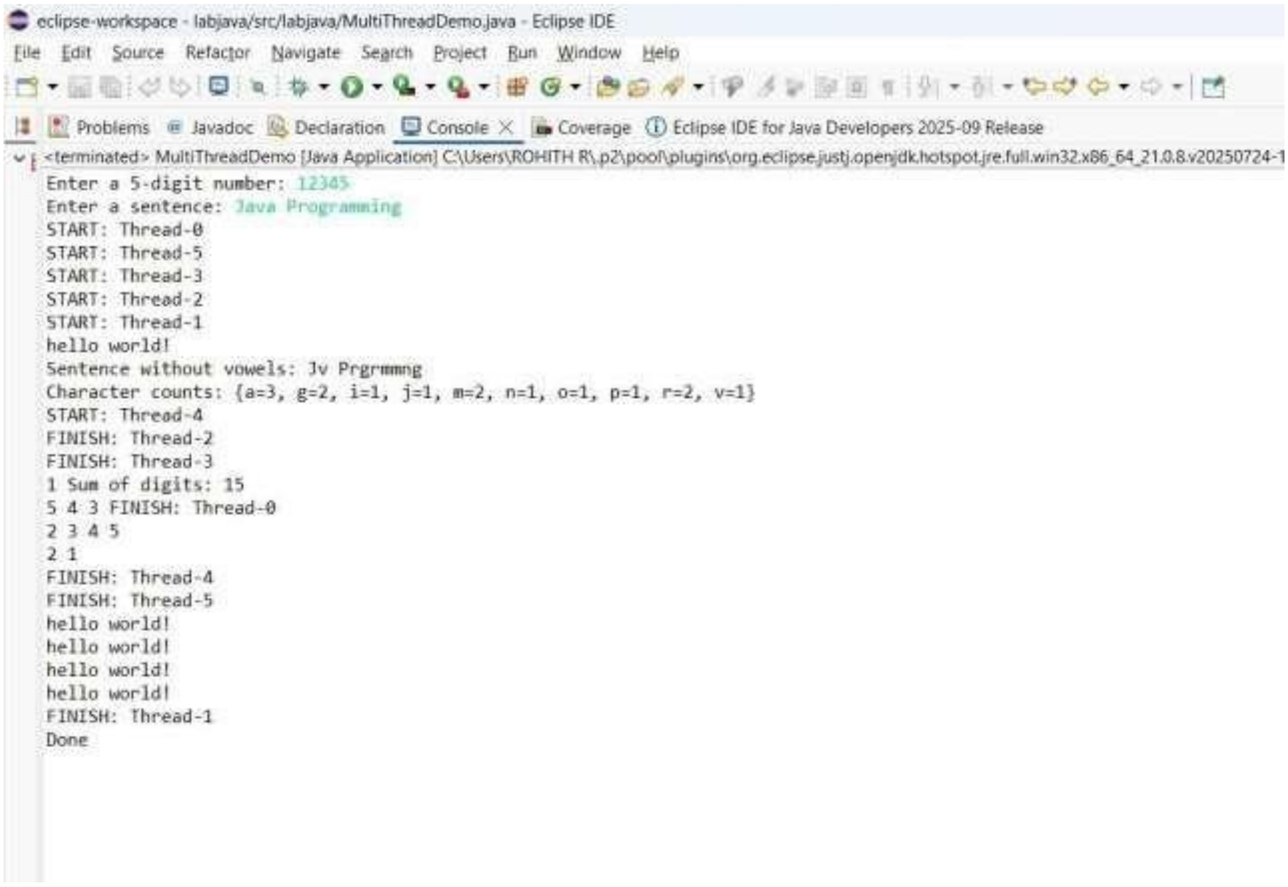
CODE:

```
package labjava;

import java.util.*;

class SumOfDigitsThread extends Thread
{ private int number;

public SumOfDigitsThread(int number)
{ this.number = number;
}

public void run()
{ System.out.println("START: " + getName());
int sum = 0, n = number;
while (n > 0)
{ sum += n %
10; n /= 10;
}
```

```
System.out.println("Sum of digits: " + sum);
System.out.println("FINISH: " + getName());
}
}

class HelloWorldThread extends Thread
{ public void run()
{ System.out.println("START: " + getName());
try {
for (int i = 0; i < 5; i++)
{ System.out.println("hello
world!"); Thread.sleep(300); //
pause briefly
}
} catch (InterruptedException e)
{ System.out.println(e);
}
System.out.println("FINISH: " + getName());
}
}

class RemoveVowelsThread extends Thread
{ private String sentence;

public RemoveVowelsThread(String sentence)
{ this.sentence = sentence;
}

public void run()
{ System.out.println("START: " + getName());
String result = sentence.replaceAll("(?i)[aeiou]", "");
System.out.println("Sentence without vowels: " + result);
System.out.println("FINISH: " + getName());
}
}

class CharCountThread extends Thread
{ private String sentence;

public CharCountThread(String sentence)
{ this.sentence = sentence;
}

public void run()
{ System.out.println("START: " + getName());
Map<Character, Integer> countMap = new TreeMap<>();
for (char c : sentence.toLowerCase().toCharArray()) {
if (c != ' ') {
countMap.put(c, countMap.getOrDefault(c, 0) + 1);
}
}
System.out.println("Character counts: " + countMap);
System.out.println("FINISH: " + getName());
}
}

class AscendingThread extends Thread
{ public void run()
{ System.out.println("START: " +
getName()); for (int i = 1; i <= 5; i++)
{  System.out.print(i + " ");
}
System.out.println();
System.out.println("FINISH: " + getName());
}
}

class DescendingThread extends Thread
{ public void run()
{ System.out.println("START: " + getName());
for (int i = 5; i >= 1; i--) {
System.out.print(i + " ");
}
System.out.println();
System.out.println("FINISH: " + getName());
}
}

public class MultiThreadDemo
{ public static void main(String[] args)
{ Scanner sc = new
Scanner(System.in);

System.out.print("Enter a 5-digit number: ");
int number = sc.nextInt();
sc.nextLine(); // consume newline

System.out.print("Enter a sentence: ");
String sentence = sc.nextLine();

Thread t1 = new SumOfDigitsThread(number);
Thread t2 = new HelloWorldThread();
Thread t3 = new RemoveVowelsThread(sentence);
Thread t4 = new CharCountThread(sentence);
Thread t5 = new AscendingThread();
Thread t6 = new DescendingThread();

t1.start();
t2.start();
t3.start();
t4.start();
t5.start();
t6.start();
```

```
try
{ t1.join();
t2.join();
t3.join();
t4.join();
t5.join();
t6.join();
} catch (InterruptedException e)
{ System.out.println(e);
}

System.out.println("Done");
sc.close();
}
}
```

OUTPUT:


eclipse-workspace - labjava/src/labjava/MultiThreadDemo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> MultiThreadDemo [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20250724-1
Enter a 5-digit number: 12345
Enter a sentence: Java Programming
START: Thread-0
START: Thread-5
START: Thread-3
START: Thread-2
START: Thread-1
hello world!
Sentence without vowels: Jv Prgrmmng
Character counts: {a=3, g=2, i=1, j=1, m=2, n=1, o=1, p=1, r=2, v=1}
START: Thread-4
FINISH: Thread-2
FINISH: Thread-3
1 Sum of digits: 15
5 4 3 FINISH: Thread-0
2 3 4 5
2 1
FINISH: Thread-4
FINISH: Thread-5
hello world!
hello world!
hello world!
hello world!
FINISH: Thread-1
Done
```

QUESTION 2:

Write a Java program that uses two threads and a shared ArrayList to demonstrate Inter-Thread Communication (ITC).

- Requirements
- Use an ArrayList<Integer> as a shared buffer with capacity = 5.
- Create a Producer thread that adds numbers 1 to 20 to the buffer.
- Create a Consumer thread that removes and “processes” the numbers.
- When the buffer is full, the Producer must wait; when the buffer is empty, the Consumer must wait.
- Use only: synchronized, wait(), notify() / notifyAll().
- Always check full/empty conditions inside a while loop (not if).
- Print clear logs for each action (produced/consumed, buffer size).
- After all, 20 items are produced and consumed, print a final report: produced=20, consumed=20, final size=0 and “Done”.

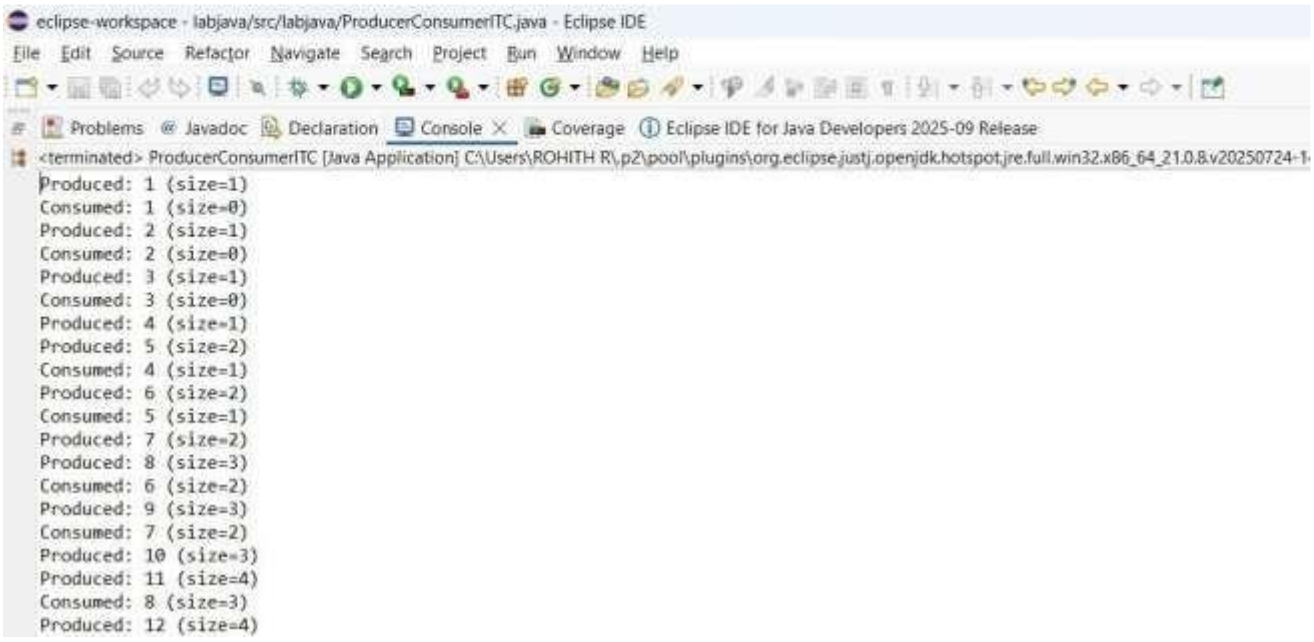
CODE:

```
 package labjava;

import java.util.ArrayList;

class SharedBuffer {
private final ArrayList<Integer> buffer = new ArrayList<>();
private final int CAPACITY = 5;
private int producedCount = 0;
private int consumedCount = 0;

public synchronized void produce(int value) throws InterruptedException
{ while (buffer.size() == CAPACITY) {
System.out.println("Producer waiting (buffer full)");
wait();
}
buffer.add(value);
producedCount++;
System.out.println("Produced: " + value + " (size=" + buffer.size() + ")");
notifyAll();
}
```

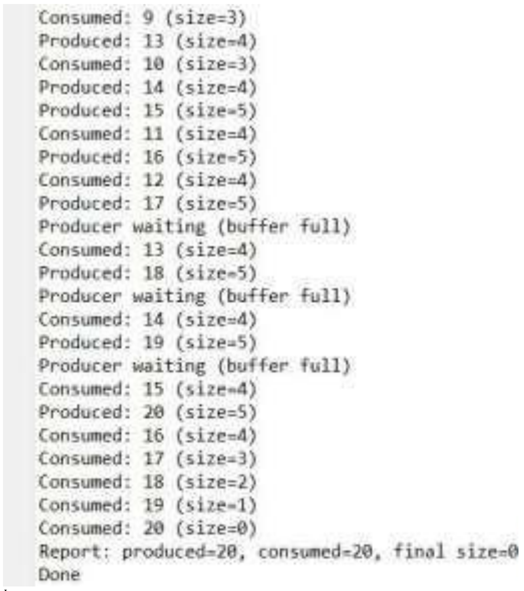
```
public synchronized int consume() throws InterruptedException
{ while (buffer.isEmpty()) {
System.out.println("Consumer waiting (buffer empty)");
wait();
}
int value = buffer.remove(0);
consumedCount++;
System.out.println("Consumed: " + value + " (size=" + buffer.size() + ")");
notifyAll();
return value;
}

public int getProducedCount()
{ return producedCount;
}

public int getConsumedCount()
{ return consumedCount;
}
public int getFinalSize()
{ return buffer.size();
}
}
class Producer extends Thread
{  private final SharedBuffer buffer;
public Producer(SharedBuffer buffer)
{ this.buffer = buffer;
}
public void run() {
for (int i = 1; i <= 20; i++)
{ try {
buffer.produce(i);
Thread.sleep(100);
} catch (InterruptedException e)
{ e.printStackTrace();
}
}
}
}
class Consumer extends Thread
{ private final SharedBuffer buffer;
public Consumer(SharedBuffer buffer)
{ this.buffer = buffer;
}
public void run() {
for (int i = 1; i <= 20; i++)
{ try {
buffer.consume();
Thread.sleep(150);
} catch (InterruptedException e)
{ e.printStackTrace();
}
}
}
}

public class ProducerConsumerITC
{ public static void main(String[] args)
{ SharedBuffer buffer = new
SharedBuffer(); Producer producer = new
Producer(buffer);
Consumer consumer = new Consumer(buffer);
producer.start();
consumer.start();
try
{ producer.join();
consumer.join();
} catch (InterruptedException e)
{ e.printStackTrace();
}
System.out.println("Report: produced=" + buffer.getProducedCount()
+ ", consumed=" + buffer.getConsumedCount()
+ ", final size=" + buffer.getFinalSize());
```

```
System.out.println("Done");
}
}
```

OUTPUT:


eclipse-workspace - labjava/src/labjava/ProducerConsumerITC.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Coverage Eclipse IDE for Java Developers 2025-09 Release

```
<terminated> ProducerConsumerITC [Java Application] C:\Users\ROHITH R\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20250724-1
Produced: 1 (size=1)
Consumed: 1 (size=0)
Produced: 2 (size=1)
Consumed: 2 (size=0)
Produced: 3 (size=1)
Consumed: 3 (size=0)
Produced: 4 (size=1)
Produced: 5 (size=2)
Consumed: 4 (size=1)
Produced: 6 (size=2)
Consumed: 5 (size=1)
Produced: 7 (size=2)
Produced: 8 (size=3)
Consumed: 6 (size=2)
Produced: 9 (size=3)
Consumed: 7 (size=2)
Produced: 10 (size=3)
Produced: 11 (size=4)
Consumed: 8 (size=3)
Produced: 12 (size=4)
```



```
Consumed: 9 (size=3)
Produced: 13 (size=4)
Consumed: 10 (size=3)
Produced: 14 (size=4)
Produced: 15 (size=5)
Consumed: 11 (size=4)
Produced: 16 (size=5)
Consumed: 12 (size=4)
Produced: 17 (size=5)
Producer waiting (buffer full)
Consumed: 13 (size=4)
Produced: 18 (size=5)
Producer waiting (buffer full)
Consumed: 14 (size=4)
Produced: 19 (size=5)
Producer waiting (buffer full)
Consumed: 15 (size=4)
Produced: 20 (size=5)
Consumed: 16 (size=4)
Consumed: 17 (size=3)
Consumed: 18 (size=2)
Consumed: 19 (size=1)
Consumed: 20 (size=0)
Report: produced=20, consumed=20, final size=0
Done
```

RESULT:

Thus the java program of using multithreading to achieve concurrent execution and improve application performance was successfully executed and output was verified.