

EXPERIMENT NO. 13

NAME – SHAIKH ARSHAD AJIJ

ROLL NO – B007

BATCH- B1

SAP ID – 60019230064

DATE – 19-4-24

AIM / OBJECTIVE:

To implement Multithreading

1. To implement Multithreading

- a. Write a multithreaded program a java program to print Table of Five, Seven and Thirteen using Multithreading (Use Thread class for the implementation).

CODE-

```
class PrintTable extends Thread {
    private int tableNumber;

    public PrintTable(int tableNumber) {
        this.tableNumber = tableNumber;
    }

    @Override
    public void run() {
        System.out.println("Table of " + tableNumber + ":");
        for (int i = 1; i <= 10; i++) {
            System.out.println(tableNumber + " * " + i + " = " + (tableNumber * i));
        }
    }
}

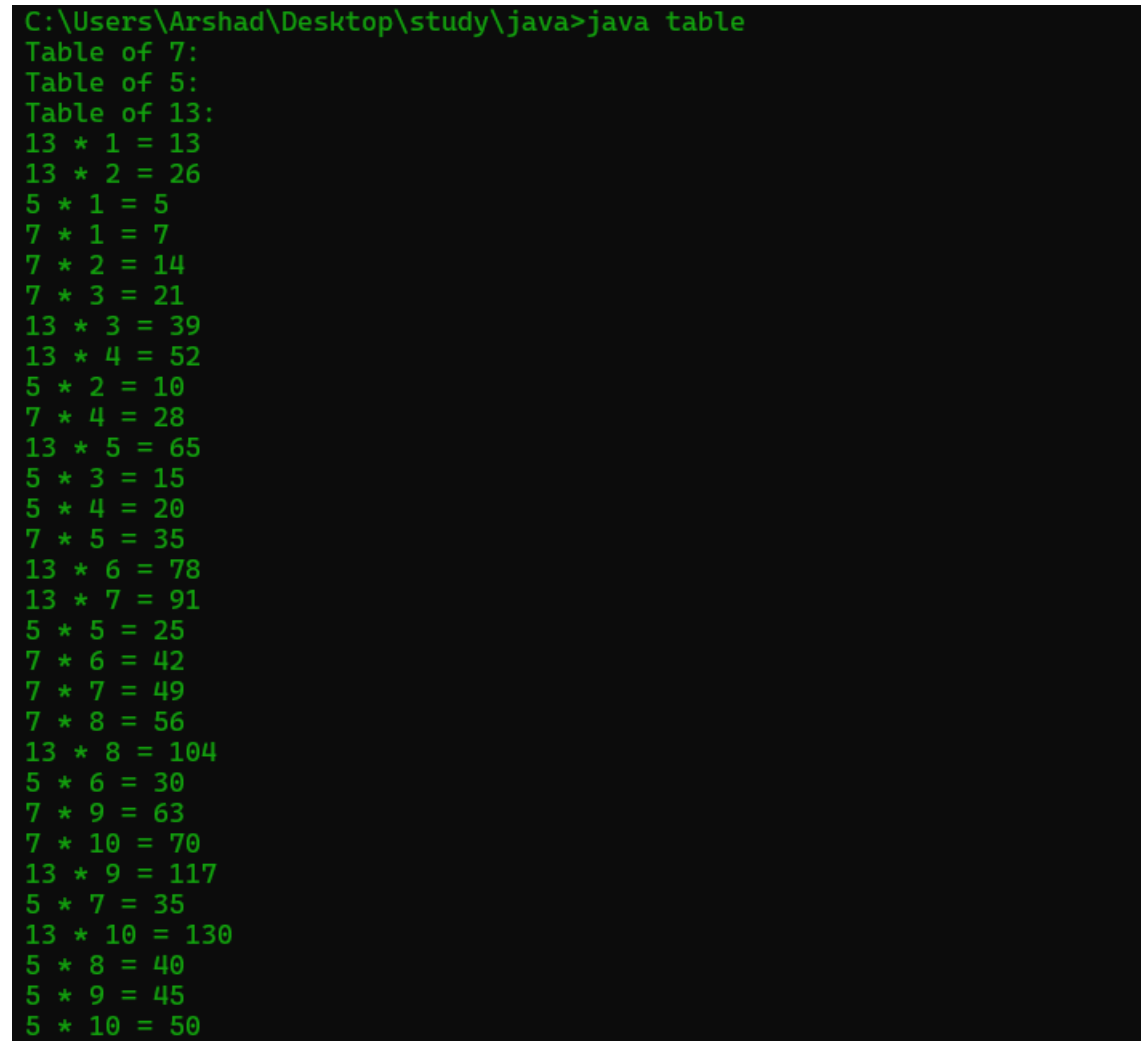
public class table {
    public static void main(String[] args) {
        PrintTable tableOfFive = new PrintTable(5);
        PrintTable tableOfSeven = new PrintTable(7);
        PrintTable tableOfThirteen = new PrintTable(13);
    }
}
```

```

        tableOfFive.start();
        tableOfSeven.start();
        tableOfThirteen.start();
    }
}

```

OUTPUT-



```

C:\Users\Arshad\Desktop\study\java>java table
Table of 7:
Table of 5:
Table of 13:
13 * 1 = 13
13 * 2 = 26
5 * 1 = 5
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
13 * 3 = 39
13 * 4 = 52
5 * 2 = 10
7 * 4 = 28
13 * 5 = 65
5 * 3 = 15
5 * 4 = 20
7 * 5 = 35
13 * 6 = 78
13 * 7 = 91
5 * 5 = 25
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
13 * 8 = 104
5 * 6 = 30
7 * 9 = 63
7 * 10 = 70
13 * 9 = 117
5 * 7 = 35
13 * 10 = 130
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

```

- b. Write a multithreaded program to display /*/*/*/*/*/*/* using 2 child threads.

CODE-

```

class PrintPattern extends Thread {
    private String pattern;

```

OUTPUT-

```
C:\Users\Arshad\Desktop\study\java>java thread  
/**/**/**/**/**/**/**/**/**/**
```

- c. Write a program to demonstrate thread methods: wait notify suspend resume join setpriority
getpriority setname getname

CODE-

```
class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
    }

    @Override
    public void run() {
        System.out.println("Thread " + getName() + " started.");
        try {
            for (int i = 0; i < 5; i++) {
                System.out.println("Thread " + getName() + " counting: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + getName() + " interrupted.");
        }
        System.out.println("Thread " + getName() + " finished.");
    }
}

public class main2 {
    public static void main(String[] args) {
        MyThread thread1 = new MyThread("Thread 1");
        MyThread thread2 = new MyThread("Thread 2");

        // Demonstrate setName and getName methods
        System.out.println("Thread 1 name: " + thread1.getName());
        System.out.println("Thread 2 name: " + thread2.getName());

        // Demonstrate setPriority and getPriority methods
        thread1.setPriority(Thread.MIN_PRIORITY);
        thread2.setPriority(Thread.MAX_PRIORITY);
        System.out.println("Thread 1 priority: " + thread1.getPriority());
        System.out.println("Thread 2 priority: " + thread2.getPriority());
    }
}
```

```

// Demonstrate join method
try {
    thread1.start();
    thread1.join(); // Wait for thread1 to finish before proceeding
    thread2.start();
} catch (InterruptedException e) {
    e.printStackTrace();
}

// Demonstrate suspend and resume methods
thread2.suspend(); // Suspend thread2
System.out.println("Thread 2 suspended.");
try {
    Thread.sleep(2000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
thread2.resume(); // Resume thread2
System.out.println("Thread 2 resumed.");
}
}

```

OUTPUT-

```

C:\Users\Arshad\Desktop\study\java>java main2
Thread 1 name: Thread 1
Thread 2 name: Thread 2
Thread 1 priority: 1
Thread 2 priority: 10
Thread Thread 1 started.
Thread Thread 1 counting: 0
Thread Thread 1 counting: 1
Thread Thread 1 counting: 2
Thread Thread 1 counting: 3
Thread Thread 1 counting: 4
Thread Thread 1 finished.
Thread Thread 2 started.
Exception in thread "main" Thread Thread 2 counting: 0
java.lang.UnsupportedOperationException
    at java.base/java.lang.Thread.suspend(Thread.java:1826)
    at main2.main(main2.java:46)
Thread Thread 2 counting: 1
Thread Thread 2 counting: 2
Thread Thread 2 counting: 3
Thread Thread 2 counting: 4
Thread Thread 2 finished.

C:\Users\Arshad\Desktop\study\java>

```

- d. Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: create a class Input that reads the number of Fibonacci numbers that the program is to generate. The class will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in an array. When the thread finishes execution, the parent thread (Input class) will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish.

CODE-

```
import java.util.Scanner;

class FibonacciGenerator extends Thread {
    private int[] fibonacciSequence;
    private int count;

    public FibonacciGenerator(int count) {
        this.count = count;
        fibonacciSequence = new int[count];
    }

    @Override
    public void run() {
        fibonacciSequence[0] = 0;
        fibonacciSequence[1] = 1;
        for (int i = 2; i < count; i++) {
            fibonacciSequence[i] = fibonacciSequence[i - 1] + fibonacciSequence[i - 2];
        }
    }

    public int[] getFibonacciSequence() {
        return fibonacciSequence;
    }
}

public class FIBO {
```

```

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter the number of Fibonacci numbers to generate: ");
    int count = scanner.nextInt();

    FibonacciGenerator fibonacciThread = new FibonacciGenerator(count);
    fibonacciThread.start();

    try {
        fibonacciThread.join(); // Wait for fibonacciThread to finish execution
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    int[] fibonacciSequence = fibonacciThread.getFibonacciSequence();

    System.out.println("Generated Fibonacci sequence:");
    for (int i = 0; i < count; i++) {
        System.out.print(fibonacciSequence[i] + " ");
    }
}

```

OUTPUT-

```

C:\Users\Arshad\Desktop\study\java>java FIB0
Enter the number of Fibonacci numbers to generate: 7
Generated Fibonacci sequence:
0 1 1 2 3 5 8
C:\Users\Arshad\Desktop\study\java>

```

- e. WAP to prevent concurrent booking of a ticket using the concept of thread synchronization.

CODE-

```
class TicketBookingSystem {
    private int availableTickets;

    public TicketBookingSystem(int totalTickets) {
        this.availableTickets = totalTickets;
    }

    public synchronized boolean bookTicket(int numTickets) {
        if (numTickets > 0 && numTickets <= availableTickets) {
            System.out.println(Thread.currentThread().getName() + " is booking " + numTickets + "
ticket(s).");
            availableTickets -= numTickets;
            System.out.println(Thread.currentThread().getName() + " successfully booked " +
numTickets + " ticket(s).");
            System.out.println("Available tickets: " + availableTickets);
            return true;
        } else {
            System.out.println(Thread.currentThread().getName() + " failed to book tickets.");
            return false;
        }
    }
}

class BookingThread extends Thread {
    private TicketBookingSystem bookingSystem;
    private int numTickets;

    public BookingThread(TicketBookingSystem bookingSystem, int numTickets) {
```



```

        this.bookingSystem = bookingSystem;

        this.numTickets = numTickets;
    }

    @Override
    public void run() {
        bookingSystem.bookTicket(numTickets);
    }
}

public class Main3 {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem(5);

        BookingThread thread1 = new BookingThread(bookingSystem, 2);
        BookingThread thread2 = new BookingThread(bookingSystem, 3);
        BookingThread thread3 = new BookingThread(bookingSystem, 4);

        thread1.start();
        thread2.start();
        thread3.start();
    }
}

```

OUTPUT-

```

C:\Users\Arshad\Desktop\study\java>java Main3
Thread-0 is booking 2 ticket(s).
Thread-0 successfully booked 2 ticket(s).
Available tickets: 3
Thread-2 failed to book tickets.
Thread-1 is booking 3 ticket(s).
Thread-1 successfully booked 3 ticket(s).
Available tickets: 0

```

CONCLUSION:

I learned about multithreading in Java. I covered thread creation, synchronization, lifecycle, and method and implemented various multithreaded programs, including printing tables, displaying patterns, generating Fibonacci sequences, and preventing concurrent booking of tickets.