QB:148:-
Write an application that creates and starts three threads. Each thread is instantiated from the same class. It executes a loop with 10 iterations. Each iteration displays string "HELLO",sleeps for 300 milliseconds. The application waits for all the threads to complete & displays the message "Good Bye..."

```
public class Qb148
{
    public static void main(String[] args)
    {
        // Create and start three threads
        Thread thread1 = new Thread(new HelloThread());
        Thread thread2 = new Thread(new HelloThread());
        Thread thread3 = new Thread(new HelloThread());

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

        try
        {
            // Wait for all threads to complete
            thread1.join();
            thread2.join();
            thread3.join();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        // All threads have completed
        System.out.println("Good Bye...");
    }
}
class HelloThread implements Runnable
{
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println("HELLO");
            try
            {
                Thread.sleep(300);
            } catch (InterruptedException e)
            {
```

```
                e.printStackTrace();
            }
        }
    }
}
```

QB:149:-

Write a program to create a child thread to print integer numbers 1 to 10

```
public class QB149
{
    public static void main(String[] args)
    {
        // Create a new thread
        Thread childThread = new Thread(new ChildRunnable());

        // Start the thread
        childThread.start();
    }
}
class ChildRunnable implements Runnable
{
    public void run()
    {
        // Print numbers from 1 to 10
        for (int i = 1; i <= 10; i++)
        {
            System.out.println(i);
        }
    }
}
```

QB:150:-

Write a program to create two threads, one thread will check whether given number is prime or not and second thread will print prime numbers between 0 to 100

```
import java.util.Scanner;

class PrimeChecker extends Thread
{
    int number;
```

```
    public PrimeChecker(int number)
    {
        this.number=number;
    }

    public void run()
    {
        int count=0;
        for(int i=1;i<=number;i++)
        {
            if(number%i==0)
            {
                count++;
            }
            else
            {
                continue;
            }
        }
        if(count==2)
        {
            System.out.println(number + ": is Prime Number");
        }
        else
        {
            System.out.println(number + ": is Not a Prime Number");
        }
    }

}
class PrimePrinter extends Thread
{
    public void run()
    {
        System.out.println("Prime Numbers Between 0 to 100 are:");
        for(int i=1; i<=100 ;i++)
        {
            int count=0;
            for(int j=1;j<=i;j++)
            {
                if(i%j==0)
                {
                    count++;
                }
                else
                {
                    continue;
```

```
                }
            }
            if(count==2)
            {
                System.out.print(i+" ");
            }
        }

    }

}
public class Qb150
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Number:");
        int number=sc.nextInt();
        PrimeChecker C=new PrimeChecker(number);
        PrimePrinter P=new PrimePrinter();
        C.start();
        try
        {
            C.join();
        }
        catch (InterruptedException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        P.start();;
    }
}
```

QB:151:-

It's IPL 2023 time. In this process of IPL-23 there is a thread called Bowler.This Bowler is going to create two threads with name – “Yorker” and “googly”.Write a java program to perform following task with each thread.

Task 1 : if the thread name is "yorker" then it should print "SIXER"

Task 2 : if the thread name is "googly" then it should print "WICKET"

Make sure first thread bowl must be yorker. When execution of first thread is completedthen and only then second bowl googly should work.

Hint: Here Bowler is a Thread Class. and It must have execution of SIXER AND WICKET. Bothyorker and googly must be from Bowler Thread Only.Give name IPL to the class of Main method.

```
class Bowler extends Thread
{
    public Bowler(String ballType)
    {
        super(ballType);
    }

    public void run()
    {
        Thread t=Thread.currentThread();
        String ballType=t.getName();
        if (ballType.equals("yorker"))
        {
            System.out.println("SIXER");
        }
        else if (ballType.equals("googly"))
        {
            System.out.println("WICKET");
        }
    }
}

class IPL
{
    public static void main(String[] args)
     {
        Bowler yorker = new Bowler("yorker");
        Bowler googly = new Bowler("googly");

        yorker.start(); // First bowl the yorker
        try
        {
            yorker.join(); // Wait for the yorker to complete
        } catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        googly.start(); // Then bowl the googly
    }
}
```

QB:152:-

Write an application that executes two threads. One thread displays "Good Morning" every 1000 milliseconds & another thread displays "Good Afternoon" every 3000 milliseconds for 10 times.. Create the threads by implementing the Runnable interface.

```
class MorningTask implements Runnable
{
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println("Good Morning");
            try
            {
                Thread.sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
class AfternoonTask implements Runnable
{
    public void run()
    {
        for (int i = 0; i < 10; i++)
        {
            System.out.println("Good Afternoon");
            try
            {
                Thread.sleep(3000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}

public class Qb151
{
    public static void main(String[] args)
    {
        Thread morningThread = new Thread(new MorningTask());
        Thread afternoonThread = new Thread(new AfternoonTask());

        morningThread.start();
        afternoonThread.start();
    }
```

```
}
```

QB:153:-
Write a program to create two threads, one thread will print odd numbers and second thread will print even numbers between 1 to 100 numbers

```
class OddThread extends Thread
{
    public void run()
    {
        for (int i = 1; i <= 100; i += 2)
        {
            System.out.println(i);
        }
    }
}
class EvenThread extends Thread
{
    public void run()
    {
        for (int i = 2; i <= 100; i += 2)
        {
            System.out.println(i);
        }
    }
}
public class QB153
{
    public static void main(String[] args)
    {
        Thread oddThread = new OddThread();
        Thread evenThread = new EvenThread();

        oddThread.start();
        evenThread.start();
    }
}
```

QB:154:-Write a complete multi-threaded program to meet following requirements:

- **Two threads of same type are to be instantiated in the method main.**
- **Each thread acts as a producer as well as a consumer.**
- **A shared buffer can store only one integer information along with the source & destination of the information at a time.**
- **The information produced is to be consumed by appropriate consumer.**
- **Both producers produce information for both consumers.**
- **Each thread produces 5 information.**

```
class ProCon
```

```
{
    int data;

    synchronized void produce(int i)
    {
            data=i;
            System.out.println("Produce="+data);
    }
    synchronized void consume()
    {
        System.out.println("consume="+data);
    }

}
class ProduceConsume extends Thread
{
    ProCon p;
    ProduceConsume(ProCon p)
    {
        this.p=p;
    }
    public void run()
    {
        System.out.println(Thread.currentThread().getName());

        for(int i=1;i<=5;i++)
        {
            p.produce(i);
            p.consume();
        }
    }
}
class QB154
{
    public static void main(String[] args)
    {
        ProCon p=new ProCon();
        ProduceConsume pc1=new ProduceConsume(p);
        ProduceConsume pc2=new ProduceConsume(p);
        pc1.start();
        try {
            pc1.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        pc2.start();
```

```
    }
}
```

QB:155:-

Create PaymentThread class by extending Thread and Make a program to count payments of 10 tickets each of 100 rs. Display total using main thread.

```
class PaymentThread extends Thread
{
    static int totalAmount = 0;
    int TICKET_PRICE = 100;
    int numTickets;

    public PaymentThread(int numTickets)
    {
        this.numTickets = numTickets;
    }

    public void run()
    {
        int payment = TICKET_PRICE * numTickets;
        totalAmount += payment;
        Thread t=Thread.currentThread();
       System.out.println(t.getName() + " collected Rs. " + payment);
    }
}

public class QB155
{
    public static void main(String[] args)
    {
        int numTicketsPerTransaction = 10;
        int numTransactions = 10;

        for (int i = 0; i < numTransactions; i++)
        {
      PaymentThread pt = new PaymentThread(numTicketsPerTransaction);
            pt.start();
            try
            {
```

```
                pt.join();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
            System.out.println("Total amount collected: Rs. " +
PaymentThread.totalAmount);
    }
}
```

QB:156:-

Write a multithreaded program to print all odd positive numbers in ascending order up to n, where n is a positive integer number given as a command line argument. Instantiate requited number of threads, where each thread except the last, examines next 50 numbers and the last thread examines remaining numbers up to n

```
class OddNumberThread extends Thread
{
    private int start;
    private int end;

    public OddNumberThread(int start, int end)
    {
        this.start = start;
        this.end = end;
    }

    public void run()
    {
        Thread t=Thread.currentThread();
        System.out.println(t.getName());
        for (int i = start; i <= end; i += 2)
        {
            System.out.println(i);
        }
    }
}

public class QB156
{
    public static void main(String[] args)
    {
```

```
        if (args.length != 1)
        {
            System.out.println("Enter Value of n in command Line");
            return;
        }

        int n = Integer.parseInt(args[0]);
        int numThreads = (n + 49) / 50; // Number of threads needed

        for (int i = 0; i < numThreads; i++)
        {
            int start = i * 50 + 1;
            int end = Math.min(start + 49, n);
           // Last thread handles remaining numbers up to n
            OddNumberThread od=new OddNumberThread(start, end);
            od.start();
            try
            {
                od.join();
            } catch (InterruptedException e)
            {
                // TODO Auto-generated catch block
            }
        }
    }
}
```

QB:157:-

Write an application that read limit from user and executes two threads. One thread displays total of first n even numbers & another thread displays total of first n odd numbers. Create the threads by implementing the Runnable interface.

```
import java.util.Scanner;

class EvenSum implements Runnable
{
    private int limit;

    public EvenSum(int limit)
    {
        this.limit = limit;
    }

    public void run()
    {
        int sum = 0;
```

```
        for (int i = 2; i <= 2 * limit; i += 2)
        {
            sum += i;
        }
       System.out.println("Sum of first " + limit + " even numbers: " + sum);
    }
}

class OddSum implements Runnable
{
    private int limit;

    public OddSum(int limit)
    {
        this.limit = limit;
    }

    public void run()
    {
        int sum = 0;
        for (int i = 1; i <= 2 * limit - 1; i += 2)
        {
            sum += i;
        }
       System.out.println("Sum of first " + limit + " odd numbers: " + sum);
    }
}
public class QB157
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the limit: ");
        int limit = scanner.nextInt();

        EvenSum evenSum = new EvenSum(limit);
        OddSum oddSum = new OddSum(limit);

        Thread evenThread = new Thread(evenSum);
        Thread oddThread = new Thread(oddSum);

        evenThread.start();
        oddThread.start();

        try
         {
            evenThread.join();
```

```
            oddThread.join();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

    }
}
```

QB:158:-

Write a program to create two thread one display alphabet from a to z and other will displaynumbers from 1 to 100

```
class AlphabetThread extends Thread
{
    public void run()
    {
        for (char c = 'a'; c <= 'z'; c++)
        {
            System.out.print(c + " ");
        }
    }
}

class NumberThread extends Thread {
    public void run()
    {
        for (int i = 1; i <= 100; i++)
        {
            System.out.print(i + " ");
        }
    }
}

public class QB158
{
    public static void main(String[] args)
    {
        AlphabetThread alphabetThread = new AlphabetThread();
        NumberThread numberThread = new NumberThread();

        alphabetThread.start();
        numberThread.start();
    }
}
```

QB:159:-
write a program that demonstrates the prodecer consumer concept in detail

```
class ProCon
{
    boolean isMarked=false;
    int data;
    synchronized void produce(int n)
    {

        if(isMarked)
        {
            try
            {
                System.out.println("Producer waiting");
                wait();
            }
            catch (Exception e)
            {
                // TODO: handle exception
            }
        }
        data=n;
        System.out.println("produce= "+data);
        isMarked=true;
        notify();
    }
    synchronized void consume()
    {

        if(!isMarked)
        {
            try
            {
                System.out.println("Consumer waiting");

                wait();
            }
            catch (Exception e)
            {
                // TODO: handle exception
            }
        }
        System.out.println("consume= "+data);
        isMarked=false;
        notify();
    }
```

```
}
class Producer extends Thread
{
    ProCon pc;
    Producer(ProCon pc)
    {
        this.pc=pc;
        //start();
    }
    public void run()
    {
        int i;
        for(i=1;i<=5;i++)
        {
            pc.produce(i);
        }
    }
}
class Consumer extends Thread
{
   ProCon pc;
    Consumer(ProCon pc)
    {
        this.pc=pc;
        //start();
    }
    public void run()
    {
        int i;
        for(i=1;i<=5;i++)
        {
            pc.consume();
        }
    }
}
class QB159
{
    public static void main(String[] args)
    {
        ProCon pc1=new ProCon();
        Producer p=new Producer(pc1);
        Consumer c=new Consumer(pc1);
        p.start();
        c.start();
    }
}
```

QB:161:-

Write a complete multi-threaded program to meet following requirements:
- Read matrix [A] m x n
- Create m number of threads
- Each thread computes summation of elements of one row, i.e. ith row of the matrix isprocessed by ith thread. Where 0 <= i< m.
- Print the results

```
import java.util.Scanner;

class RowSumThread extends Thread
{
    private int[] row;
    private int sum;

    public RowSumThread(int[] row)
    {
        this.row = row;
        this.sum = 0;
    }

    public void run()
    {
        for (int num : row)
        {
            sum += num;
        }
    }

    public int getSum()
    {
        return sum;
    }
}

public class QB161
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        // Read matrix dimensions
        System.out.print("Enter number of rows (m): ");
        int m = scanner.nextInt();
        System.out.print("Enter number of columns (n): ");
```

```
        int n = scanner.nextInt();

        int[][] matrix = new int[m][n];

        // Read matrix elements
        System.out.println("Enter matrix elements:");
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                System.out.print("Enter Element a["+i+"]["+j+"]=");
                matrix[i][j] = scanner.nextInt();
            }
        }

        // Create threads
        RowSumThread[] threads = new RowSumThread[m];
        for (int i = 0; i < m; i++)
        {
            threads[i] = new RowSumThread(matrix[i]);
            threads[i].start();
        }

        // Wait for threads to finish and print results
        try
        {
            for (int i = 0; i < m; i++)
            {
                threads[i].join();
System.out.println("Sum of elements in row "+(i+1)+": "+threads[i].getSum());
            }
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```