1. Write a Java program to perform a runnable interface, take two threads t1 and t2 and fetch the names of the thread using getName() method.

```
public class ThreadNameExample {
 public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable(), "Thread-1");
        Thread t2 = new Thread(new MyRunnable(), "Thread-2");
       t1.start();
        t2.start();
        System.out.println("Name of t1: " + t1.getName());
        System.out.println("Name of t2: " + t2.getName());
    }
    static class MyRunnable implements Runnable {
        @Override
        public void run() {
            System.out.println("Thread is running: " + Thread.currentThread().getName());
        }
    }
}
```

2.Given an integer N, the task is to write program to print the first N natural numbers in increasing order using two threads.

Input: N = 10
Output: 1 2 3 4 5 6 7 8 9 10

Input: N = 18
Output: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

```
import java.util.Scanner;

public class PrintNumbers {

 public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

       System.out.print("Enter the value of N: ");

        int N = scanner.nextInt();

       SharedPrinter printer = new SharedPrinter();

        Thread t1 = new Thread(new NumberPrinter(printer, 1, N / 2), "Thread-1");

        Thread t2 = new Thread(new NumberPrinter(printer, N / 2 + 1, N), "Thread-2");

        t1.start();

        t2.start();

       scanner.close();

    }

    static class SharedPrinter {

        private int number = 1;

         public void print(int num) {

            synchronized (this) {

                while (number <= num) {

                    while (number < num) {

                        try {
```

```
                        wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                if (number <= num) {
                    System.out.println(Thread.currentThread().getName() + ": " + number);
                    number++;
                    notifyAll();
                }
            }
        }
    }
  }
static class NumberPrinter implements Runnable {
    private final SharedPrinter printer;
    private final int start;
    private final int end;
public NumberPrinter(SharedPrinter printer, int start, int end) {
        this.printer = printer;
        this.start = start;
        this.end = end;
    }
   @Override
    public void run() {
            for (int i = start; i <= end; i++) {
                printer.print(i);
            }
        }
    }
}
```

3. Write a two-threaded program, where one thread finds all prime numbers (in 0 to 10) and another thread finds all palindrome numbers (in 10 to 50). Schedule these threads in a sequential manner to get the results.

Palindrome numbers from 10 to 50 : 11 22 33 44
Prime numbers from 0 to 10 : 2 3 5 7

```
import java.util.Scanner;

public class NumberFinder {
 public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the start of prime number range (e.g., 0): ");
        int primeStart = scanner.nextInt();
        System.out.print("Enter the end of prime number range (e.g., 10): ");
        int primeEnd = scanner.nextInt();
        System.out.print("Enter the start of palindrome number range (e.g., 10): ");
        int palindromeStart = scanner.nextInt();
        System.out.print("Enter the end of palindrome number range (e.g., 50): ");
        int palindromeEnd = scanner.nextInt();
        PrimeNumberFinder primeFinder = new PrimeNumberFinder(primeStart, primeEnd);
        PalindromeNumberFinder palindromeFinder = new
PalindromeNumberFinder(palindromeStart, palindromeEnd);
```

}

```
        Thread primeThread = new Thread(primeFinder);
        Thread palindromeThread = new Thread(palindromeFinder);

        primeThread.start();
        try {
            primeThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        palindromeThread.start();
        try {
            palindromeThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Prime numbers from " + primeStart + " to " + primeEnd + " : " +
primeFinder.getPrimeNumbers());
        System.out.println("Palindrome numbers from " + palindromeStart + " to " +
palindromeEnd + " : " + palindromeFinder.getPalindromeNumbers());

        scanner.close();
    }
 static class PrimeNumberFinder implements Runnable {
        private final StringBuilder primeNumbers = new StringBuilder();

        private final int start;

        private final int end;

            public PrimeNumberFinder(int start, int end) {

            this.start = start;

            this.end = end;

        }

      @Override

        public void run() {

            for (int i = start; i <= end; i++) {

                boolean isPrime = true;

                for (int j = 2; j <= Math.sqrt(i); j++) {

                    if (i % j == 0) {

                        isPrime = false;

                        break;

                    }

                }

                if (isPrime) {

                    primeNumbers.append(i).append(" ");

                }

            }

        }

           public String getPrimeNumbers() {

            return primeNumbers.toString().trim();

        }
```

```
}
 static class PalindromeNumberFinder implements Runnable {
  private final StringBuilder palindromeNumbers = new StringBuilder();
  private final int start;
  private final int end;
  public PalindromeNumberFinder(int start, int end) {
      this.start = start;
      this.end = end;
  }
 @Override
  public void run() {
      for (int i = start; i <= end; i++) {
          if (isPalindrome(i)) {
              palindromeNumbers.append(i).append(" ");
          }
      }
  }
      private boolean isPalindrome(int num) {
      int originalNum = num;
      int reverseNum = 0;
      while (num != 0) {
          int remainder = num % 10;
          reverseNum = reverseNum * 10 + remainder;
          num /= 10;
      }
```

```
            return originalNum == reverseNum;
        }
            public String getPalindromeNumbers() {
            return palindromeNumbers.toString().trim();
        }
    }
}
```