Name:	**Debjit Ghosal**
UID:	**2023300065**
Experiment No.	**10B**

AIM:	**Multithreading**
Program 1	
PROBLEM STATEMENT :	***Write a two-threaded program, where one thread finds all prime numbers (in 0 to 100) and another thread finds all palindrome numbers (in 10 to 1000). Schedule these threads in a sequential manner to get the results. Now reschedule them as parallel threads.***

PROGRAM:	**/*
	Write a java program for two-threaded program, where one thread finds all prime numbers (in 0 to 100) and another thread finds all palindrome numbers (in 10 to 1000). Schedule these threads in a sequential manner to get the results. Now reschedule them as parallel threads.
	*/**
	
	**import java.util.Scanner;
	import java.util.ArrayList;
	import java.util.List;**
	
	**class PrimeFinder extends Thread {
	  private List<Integer> primeNumbers = new ArrayList<>();**
	
	** @Override
	  public void run() {
	  for (int i = 0; i <= 100; i++) {
	  if (isPrime(i)) {
	  primeNumbers.add(i);
	  }
	  }
	  }**
	
	** public List<Integer> getPrimeNumbers() {**

```
        return primeNumbers;
    }

    private boolean isPrime(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }
}

class PalindromeFinder extends Thread {
    private List<Integer> palindromeNumbers = new ArrayList<>();

    @Override
    public void run() {
        for (int i = 10; i <= 1000; i++) {
            if (isPalindrome(i)) {
                palindromeNumbers.add(i);
            }
        }
    }

    public List<Integer> getPalindromeNumbers() {
        return palindromeNumbers;
    }

    private boolean isPalindrome(int num) {
        int temp = num;
        int reverse = 0;
        while (temp != 0) {
            int digit = temp % 10;
            reverse = reverse * 10 + digit;
            temp /= 10;
        }
        return num == reverse;
    }
}
```

```
public class TwoThread {
   public static void main(String[] args) {
      // Sequential Execution
      PrimeFinder primeFinder = new PrimeFinder();
      PalindromeFinder palindromeFinder = new PalindromeFinder();

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

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

      System.out.println("The Sequential Prime Numbers: " +
primeFinder.getPrimeNumbers());
      System.out.println("The Sequential Palindrome Numbers: " +
palindromeFinder.getPalindromeNumbers());

      // Parallel Execution
      PrimeFinder parallelPrimeFinder = new PrimeFinder();
      PalindromeFinder parallelPalindromeFinder = new
PalindromeFinder();

      parallelPrimeFinder.start();
      parallelPalindromeFinder.start();
  try {
         parallelPrimeFinder.join();
         parallelPalindromeFinder.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
            }
        }
      }
```

RESULT:

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac TwoThread.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java TwoThread
Sequential :
Prime Numbers: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
Palindrome Numbers: 11 22 33 44 55 66 77 88 99 101 111 121 131 141 151 161 171 1
81 191 202 212 222 232 242 252 262 272 282 292 303 313 323 333 343 353 363 373 3
83 393 404 414 424 434 444 454 464 474 484 494 505 515 525 535 545 555 565 575 5
85 595 606 616 626 636 646 656 666 676 686 696 707 717 727 737 747 757 767 777 7
87 797 808 818 828 838 848 858 868 878 888 898 909 919 929 939 949 959 969 979 9
89 999

Parallel :
Prime Numbers: 2 3 5 7 11 13 17 19 23 29 31 37 Palindrome Numbers: 41 43 47 11 5
3 22 59 33 61 44 67 55 71 66 73 77 79 88 83 99 89 101 97
111 121 131 141 151 161 171 181 191 202 212 222 232 242 252 262 272 282 292 303
313 323 333 343 353 363 373 383 393 404 414 424 434 444 454 464 474 484 494 505
515 525 535 545 555 565 575 585 595 606 616 626 636 646 656 666 676 686 696 707
717 727 737 747 757 767 777 787 797 808 818 828 838 848 858 868 878 888 898 909
919 929 939 949 959 969 979 989 999
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac TwoThread.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java TwoThread
Sequential Prime Numbers: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53,
 59, 61, 67, 71, 73, 79, 83, 89, 97]
Sequential Palindrome Numbers: [11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121, 131
, 141, 151, 161, 171, 181, 191, 202, 212, 222, 232, 242, 252, 262, 272, 282, 292, 303,
 313, 323, 333, 343, 353, 363, 373, 383, 393, 404, 414, 424, 434, 444, 454, 464, 474,
484, 494, 505, 515, 525, 535, 545, 555, 565, 575, 585, 595, 606, 616, 626, 636, 646, 6
56, 666, 676, 686, 696, 707, 717, 727, 737, 747, 757, 767, 777, 787, 797, 808, 818, 82
8, 838, 848, 858, 868, 878, 888, 898, 909, 919, 929, 939, 949, 959, 969, 979, 989, 999
]
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

Program 2	
PROBLEM STATEMENT :	**Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.** ***Sample Output:***

	Random Integer generated : 82 Square of 82 = 6724 Cube of 82 = 551368

PROGRAM:	**/*
	Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 2 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.**
	
	Sample Output:
	
	Random Integer generated : 82
	
	Square of 82 = 6724
	
	**Cube of 82 = 551368
	*/**
	
	**import java.util.Scanner;
	import java.util.Random;**
	
	**class RandomNumberGenerator extends Thread {
	  private Random random = new Random();**
	
	** @Override
	  public void run() {
	  for (int i = 0; i < 1; i++) {
	  int randomNumber = random.nextInt(100);
	  System.out.println("Random Integer generated: " +
	randomNumber);
	  try {
	  Thread.sleep(2000);
	  } catch (InterruptedException e) {
	  e.printStackTrace();
	  }
	  new SquareThread(randomNumber).start();**

```
            new CubeThread(randomNumber).start();
        }
    }
}

class SquareThread extends Thread {
    private int number;

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

    @Override
    public void run() {
        System.out.println("Square of " + number + " = " + (number *
number));
    }
}

class CubeThread extends Thread {
    private int number;

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

    @Override
    public void run() {
        System.out.println("Cube of " + number + " = " + (number *
number * number));
    }
}

public class ThreeThread {
    public static void main(String[] args) {
        new RandomNumberGenerator().start();
    }
}
```

RESULT:

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac ThreeThread.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java ThreeThread
Random Integer generated: 66
Square of 66 = 4356
Cube of 66 = 287496
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

```
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ javac ThreeThread.java
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$ java ThreeThread
Random Integer generated: 60
Random Integer generated: 3
Square of 60 = 3600
Cube of 60 = 216000
Square of 3 = 9
Random Integer generated: 8
Cube of 3 = 27
Square of 8 = 64
Cube of 8 = 512
psipl@psipl-OptiPlex-3000:~/Desktop/2023300065$
```

CONCLUSION:	I have learnt about multithreading and use of threads