Name: Aum Kulakrni

Div: D6AD

Roll No. 36

Experiment Title:
Program on Multithreading

WAJP to print the table of 5,7,13 using Multithreading (Use thread class)

Code:

```
class PrintTables extends Thread
{
  int num;
  public PrintTables(int num)
  {
    super("Table" + num);
    this.num = num;
  }
  public void run()
  {
    for(int i = 1; i <= 10; i++)
    {
      System.out.println(num + "*"+ i + ": " + num * i);
    }
  }
}
class Tables
{
  public static void main(String args[])
  {
    PrintTables  t5 = new PrintTables(5);
    PrintTables  t7 = new PrintTables(7);
    PrintTables  t13 = new PrintTables(13);

    t5.start();
    t7.start();
    t13.start();

    try
    {
      t5.join();
      t7.join();
      t13.join();
    }
    catch(InterruptedException e)
    {
      System.out.println(e);
    }
  }
}
```

Ouput:

```
5*1: 5
5*2: 10
13*1: 13
7*1: 7
13*2: 26
5*3: 15
5*4: 20
13*3: 39
13*4: 52
7*2: 14
13*5: 65
13*6: 78
13*7: 91
13*8: 104
13*9: 117
5*5: 25
13*10: 130
7*3: 21
5*6: 30
5*7: 35
7*4: 28
5*8: 40
5*9: 45
7*5: 35
7*6: 42
5*10: 50
7*7: 49
7*8: 56
7*9: 63
7*10: 70
```

WAJP to print first 20 prime nos. and 15 Fibonacci terms. (Using Runnable Interface)

Code:

```
class PrintPrimes implements Runnable
{
  Thread t;
  int noOfElements;
  public PrintPrimes(int noOfElements)
  {
    t = new Thread(this, "print primes");
    this.noOfElements = noOfElements;
    t.start();
  }
  public void run()
  {
    int i = 1;
    int num = 2;
    boolean isPrime = true;
    while( i <= noOfElements)
    {
      for(int j = 2; j <= num / 2; j++)
      {
        if( num % j == 0)
        {
          isPrime = false;
          break;
        }
      }
      if(isPrime)
      {
        System.out.println( "Prime No. " + num + "  ( "+ i +" )");
        i++;
      }
      isPrime = true;
      num++;
    }
  }
}
class PrintFibo implements Runnable
{
  Thread t;
  int noOfElements;
  public PrintFibo(int noOfElements)
  {
    t = new Thread(this, "print fibos");
```

```
43      this.noOfElements = noOfElements;
44      t.start();
45    }
46    public void run()
47    {
48      int a = 0;
49      int b = 1;
50      for(int i = 1; i <= noOfElements; i++)
51      {
52        System.out.println("Fibo term "+ i + ": " + a);
53        b = a + b;
54        a = b - a;
55      }
56    }
57  }
58
59  class PrimeAndFibo
60  {
61    public static void main(String args[])
62    {
63      PrintPrimes prPrimes = new PrintPrimes(20);
64      PrintFibo prFibos = new PrintFibo(15);
65      try
66      {
67        prPrimes.t.join();
68        prFibos.t.join();
69      }
70      catch(InterruptedException e)
71      {
72        System.out.println(e);
73      }
74
75    }
76  }
```

Ouput:

```
Prime No. 2  ( 1 )
Prime No. 3  ( 2 )
Prime No. 5  ( 3 )
Prime No. 7  ( 4 )
Prime No. 11  ( 5 )
Prime No. 13  ( 6 )
Prime No. 17  ( 7 )
Prime No. 19  ( 8 )
Prime No. 23  ( 9 )
Fibo term 1: 0
Prime No. 29  ( 10 )
Prime No. 31  ( 11 )
Prime No. 37  ( 12 )
Prime No. 41  ( 13 )
Fibo term 2: 1
Prime No. 43  ( 14 )
Prime No. 47  ( 15 )
Fibo term 3: 1
Fibo term 4: 2
Fibo term 5: 3
Prime No. 53  ( 16 )
Prime No. 59  ( 17 )
Fibo term 6: 5
Prime No. 61  ( 18 )
Prime No. 67  ( 19 )
Fibo term 7: 8
Fibo term 8: 13
Prime No. 71  ( 20 )
Fibo term 9: 21
Fibo term 10: 34
Fibo term 11: 55
Fibo term 12: 89
Fibo term 13: 144
Fibo term 14: 233
Fibo term 15: 377
```

Program on concept of synchronization

Code:

```
1
2  class PrintValue
3  {
4      public void printValue(int num, int i)
5      {
6          System.out.println(num + "*" + i + ": " + num * i);
7      }
8  }
9  class Table implements Runnable
10 {
11     PrintValue pv;
12     int num;
13     Thread t;
14     public Table(int num, PrintValue pv)
15     {
16         this.num = num;
17         this.pv = pv;
18         t = new Thread(this, "Table of " + num);
19     }
20     private void printTable()
21     {
22         for (int i = 1; i <= 10; i++)
23         {
24             synchronized(pv)
25             {
26                 pv.printValue(num, i);
27             }
28         }
29     }
30     public void run()
31     {
32         printTable();
33     }
34 }
35 class MultiThreading
36 {
37     public static void main(String args[])
38     {
39         PrintValue pv = new PrintValue();
40         Table t1 = new Table(3, pv);
41         Table t2 = new Table(5, pv);
42         Table t3 = new Table(7, pv);
43         t1.t.start();
44         t2.t.start();
```

```
45          t3.t.start();
46
47          try
48          {
49              t1.t.join();
50              t2.t.join();
51              t3.t.join();
52          }
53          catch(InterruptedException e)
54          {
55              System.out.println(e);
56          }
57      }
58  }
```

Ouput:

```
Balance: 500
Balance: 0
Cannot Withdraw
Balance: 0
Cannot Withdraw
Balance: 0
Cannot Withdraw
Balance: 500
Balance: 0
Balance: 500
Balance: 0
Balance: 0
Cannot Withdraw
Balance: 500
Balance: 500
Balance: 1000
Balance: 0
Balance: 1500
Balance: 1000
Balance: 1500
Balance: 1000
Balance: 1500
Balance: 2000
Balance: 2500
```

If we make the functions with operating on the shared data synchronized the only one thread will be able to access it at once Thus first all the withdraws will occur after all the deposits

Code:

```
1  class Account
```

```
{
    int balance;
    public Account(int balance)
    {
        this.balance = balance;
    }
    synchronized public void deposit(int value)
    {
        balance = balance + value;
    }
    synchronized public void withdraw(int value)
    {
        if(balance - value < 0)
        {
            System.out.println("Cannot Withdraw");
        }
        balance = balance - value;
    }
}
```

Ouput:

```
Balance: 1000
Balance: 1500
Balance: 2000
Balance: 2500
Balance: 3000
Balance: 3500
Balance: 4000
Balance: 4500
Balance: 5000
Balance: 5500
Balance: 5000
Balance: 4500
Balance: 4000
Balance: 3500
Balance: 3000
Balance: 2500
Balance: 2000
Balance: 1500
Balance: 1000
Balance: 500
```