ASSIGNMENT 2

NAME : HRITHVIK KONDALKAR
ROLL : 002211001088

1. Write a program to create two threads. Print “In main thread” in main thread and “In child thread” in child thread.

```
class TestThread implements Runnable{
  Thread t;
  TestThread(String name){
    t = new Thread(this, name);
    t.start();
  }
  public void run(){
    System.out.println("In "+ t.getName() + " thread");
  }
}
public class Main{
  public static void main(String[] a){
    System.out.println("In main thread");
    TestThread child = new TestThread("Child");
  }
}
```

```
[be2288@localhost 1]$ java Main1
In main Thread
In Child Thread
```

2. Create two threads and call them EvenThread and OddThread. EvenThread will print number as 2 4 6 8 10... and Odd Thread will print number as 1 3 5.... Now, synchronize these two threads to get the output as: 1 2 3 4 5 6 7 8.

```
class NumberPrinter implements Runnable {
   private static Object lock = new Object();
   private static int last_printed;
   private int start;
   private int stop;
   private int step;
   private int current;
   NumberPrinter(int start, int stop, int step) {
      this.start = start;
      this.stop = stop;
      this.step = step;
      this.current = start;
   }
   public void run() {
      while (current < stop) {
         synchronized (lock) {
            while ((current != (last_printed+1))) {
               try {
                  lock.wait();
               } catch (InterruptedException e) {
                  e.printStackTrace();
               }
            }
            System.out.print(current + " ");
            last_printed = current;
            current = current + step;
            lock.notifyAll();
         }
      }
   }
}
public class Main {
   public static void main(String[] a) {
      Thread odd = new Thread(new NumberPrinter(1, 9, 2));
      Thread even = new Thread(new NumberPrinter(2, 9, 2));
      odd.start();
      even.start();
   }
}
```

```
[be2288@localhost 2]$ java Main
1 2 3 4 5 6 7 8 [be2288@localho
```

3. Consider the following series x = 1+1/1! +1/2! + 1/3! + ...1/10! Create two threads t1 & t2. t1 will generate the denominators and t2 will form the term and add them up. Finally print the result.

ipc/Consumer.java

```
package ipc;

public class Consumer extends Thread {

  double[] buffer;
  private double currentSum;
  private int currentIteration;
  private int end;

  public Consumer(double[] buffer, int end) {
    currentSum = 0;
    this.buffer = buffer;
    currentIteration = 0;
    this.end = end;
  }

  private void consumerTask() {
    currentSum = currentSum + 1/buffer[0];
    buffer[0] = -1.0;
  }

  public void run() {
    while (currentIteration != end) {
      while (buffer[0] == -1.0){
        try{
          this.sleep(1);
        }catch(InterruptedException e){
          e.printStackTrace();
        }
      }
        currentIteration++;
        consumerTask();
    }
    System.out.println("sum of "+ end+" terms  : "+ currentSum);
  }
}
```

ipc/Producer.java

```
package ipc;

public class Producer extends Thread {
  double[] buffer;
  private double previousFactorial;
  private int currentIteration;
  private int end;
  public Producer(double[] buffer, int end) {
    this.buffer = buffer;
    this.buffer[0] = -1.0;
    this.end = end;
    currentIteration = 0;
    previousFactorial = 1.0;
  }
  private double nextFactorial() {
    previousFactorial = previousFactorial * currentIteration;
    return previousFactorial;
  }

  private void producerTask() {
    buffer[0] = nextFactorial();
  }

  public void run() {
    while (currentIteration != end) {
      while (buffer[0] != -1.0){
        try{
          this.sleep(1);
        }catch(InterruptedException e){
          e.printStackTrace();
        }
      }
      currentIteration++;
      producerTask();
    }
  }
}
```

```
import ipc.Producer;
import ipc.Consumer;

public class Main {
  public static void main(String[] a) {
    double[] buffer = new double[1];
    Thread producer = new Thread(new Producer(buffer, 10));
    Thread consumer = new Thread(new Consumer(buffer, 10));
    producer.start();
    consumer.start();
  }
}
```

```
[be2288@localhost 3]$ java Main
sum of 10 terms  : 1.7182818011463847
```

4. Consider a file that contains a number of integers. Create two threads. Call them ‘producer’ and ‘consumer’ thread. Producer thread will be reading the integers from the file continuously while consumer thread will add them up. Use proper synchronization mechanism if needed.

```
import java.io.*;

class ReaderThread implements Runnable {
   boolean[] end;
   boolean[] readerTurn;
   int[] buffer;
   BufferedReader br;

   ReaderThread(String path, boolean[] readerTurnBuffer, boolean[] endBuffer, int[] buffer) {
      try {
         br = new BufferedReader(new FileReader(path));
         end = endBuffer;
         end[0] = false;
         readerTurn = readerTurnBuffer;
         readerTurn[0] = true;
         this.buffer = buffer;
      } catch (FileNotFoundException f) {
         System.out.println("File was not found at " + path);
         end[0] = true;
      }
   }

   private int readInt() {
      try {
         String line = br.readLine();
         readerTurn[0] = true;
         if (line != null && !line.isEmpty()) {
            return Integer.parseInt(line);
         } else {
            end[0] = true;
            return 0;
         }
      } catch (IOException e) {
         e.printStackTrace();
         return 0;
      }
   }
```

```
    public synchronized void run() {
        while (!end[0]) {
            while (!readerTurn[0]) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            buffer[0] = readInt();
            readerTurn[0] = false;
            notify();
        }
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class Main {
    public static void main(String[] arg) {
        String path = "nums.txt";
        boolean[] endBuf = new boolean[1];
        boolean[] readerTurnBuf = new boolean[1];
        int[] resBuf = new int[1];
        int sum = 0;

        ReaderThread readerThread = new ReaderThread(path, readerTurnBuf, endBuf, resBuf);
        Thread child = new Thread(readerThread);
        child.start();

        synchronized (readerThread) {
            try {
                while (!endBuf[0]) {
                    while (readerTurnBuf[0]) {
                        readerThread.wait();
                    }
                    sum += resBuf[0];
                    readerTurnBuf[0] = true;
                    System.out.println("Current sum: " + sum + "  Last read: " + resBuf[0]);
                    readerThread.notify();
```

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

```
[be2288@localhost 4]$ java Main
Current sum: 12  Last read: 12
Current sum: 14  Last read: 2
Current sum: 24  Last read: 10
Current sum: 39  Last read: 15
Current sum: 39  Last read: 0
```

5. Consider the series 1+2+3+…+100. This can be considered as (1+3+5+…+99)+(2+4+6+…+100). Create two threads to compute two series in parallel (do not use simplified equation). Finally print the final sum.

```
class Adder implements Runnable{
  int current;
  int end;
  int step;
  int[] buffer;
  Adder(int start, int stop, int step, int[] buffer){
    current = start;
    end = stop;
    this.step = step;
    this.buffer = buffer;
    buffer[0] = 0;
  }
  public void run(){
    synchronized(buffer){
      while(current<end){
        buffer[0] = buffer[0]+current;
        current = current + step;
      }
    }
  }
}
class Main{
  public static void main(String[] a){
    int[] bufferOdd = new int[1];
    int[] bufferEven = new int[1];
    Thread odd = new Thread(new Adder(1,101,2,bufferOdd));
    Thread even = new Thread(new Adder(2,101,2, bufferEven));
    odd.start();
    even.start();
    try{
      odd.join();
      even.join();
      int sum = bufferOdd[0]+bufferEven[0];
      System.out.println("Sum = "+sum);
    }catch(InterruptedException e){
      e.printStackTrace();
    }
  }
}
```

```
[be2288@localhost 5]$ java Main
Sum = 5050
```

6. Consider the following parallel binary search algorithm for series a1, a2…an sorted in increasing order such that n mod 10 = 0. Element to be searched is e.
    1. Create n/10 threads t1, t2,..,tn/10.
    2. Distribute the numbers among threads such that ti will have numbers ai, ai+1, ….a2i-1.
    3. Distribute the element e to all threads.
    4. Each thread searches the element e in its sub-array using binary search algorithm.

```
import java.util.Arrays;
import java.util.Random;

class BinarySearch implements Runnable{
 int startIndex;
 int endIndex;
 int currentIndex;
 int[] array;
 int target;
 static boolean found = false;
 static int threadsExhausted = 0;
 BinarySearch(int[] searchArray,int startIndex, int endIndex, int target){
  array = searchArray;
  this.startIndex = startIndex;
  this.endIndex = endIndex;
  this.target = target;
  currentIndex = (startIndex+endIndex)/2;
 }
 private int updateCurrentIndex(){
  if(array[currentIndex]>target){
   endIndex = currentIndex;
   currentIndex = (endIndex+startIndex)/2;
  }
  else{
   startIndex = currentIndex;
   currentIndex = (endIndex+startIndex)/2;
  }
  return currentIndex;
 }
 public void run(){
  while(!found && currentIndex!=updateCurrentIndex()){
     if(array[currentIndex]==target){
      System.out.println("found "+target+" at index "+currentIndex+" by thread : "+
Thread.currentThread());
      found = true;
     }
```

```
    }
    threadsExhausted++;
  }
}

class ThreadManager{
  int lastIndex;
  int[] array;
  int target;
  ThreadManager(int[] searchArray,int target){
    lastIndex = searchArray.length;
    array = searchArray;
    Arrays.sort(array);
    this.target = target;
    for(int i = 0;i<(lastIndex+1)/10;i++){
      Thread temp = new Thread(new BinarySearch(array,i*10,i*10+9,target));
      temp.start();
    }

    while(BinarySearch.threadsExhausted!=(lastIndex+1)/10){
      try{
        Thread.currentThread().sleep(1);
      }catch(InterruptedException e){
        e.printStackTrace();
      }
    }
    if(!BinarySearch.found){
      System.out.println(target +" is not in array.");
    }
  }
}
public class Main {
    public static void main(String[] args) {
        int[] searchArray = generateRandomArray(50);
        //int target = searchArray[23];
        int target = 43;
        ThreadManager threadManager = new ThreadManager(searchArray, target);
        System.out.println("Array: " + Arrays.toString(searchArray));
    }

    private static int[] generateRandomArray(int size) {
        int[] array = new int[size];
        Random random = new Random();
        for (int i = 0; i < size; i++) {
```

```
            array[i] = random.nextInt(100);
        }
        return array;
    }
}
```

```
[be2288@localhost 6]$ java Main
43 is not in array.
Array: [6, 8, 11, 11, 11, 11, 13, 14, 15, 17, 23,
24, 27, 28, 28, 30, 30, 36, 36, 43, 53, 55, 56, 58
, 59, 62, 63, 64, 65, 67, 68, 69, 70, 71, 72, 78,
80, 83, 84, 85, 86, 87, 87, 88, 89, 90, 90, 95, 95
, 98]
```

```
[be2288@localhost 6]$ java Main
found 43 at index 26 by thread : Thread[Thread-2,5
,main]
Array: [0, 7, 10, 10, 14, 15, 17, 18, 19, 21, 22,
22, 25, 29, 32, 32, 34, 35, 35, 36, 36, 37, 37, 38
, 39, 39, 43, 52, 55, 55, 58, 61, 62, 66, 67, 67,
68, 71, 71, 73, 75, 77, 79, 83, 88, 90, 90, 93, 94
, 99]
```

7. Write a Java program using threading technology and print the thread index and location where the element has been found.

```
import java.util.Arrays;
import java.util.Random;

class BinarySearch implements Runnable{
  int startIndex;
  int endIndex;
  int currentIndex;
  int[] array;
  int target;
  static boolean found = false;
  static int threadsExhausted = 0;
  BinarySearch(int[] searchArray,int startIndex, int endIndex, int target){
    array = searchArray;
    this.startIndex = startIndex;
    this.endIndex = endIndex;
    this.target = target;
    currentIndex = (startIndex+endIndex)/2;
  }
  private int updateCurrentIndex(){
    if(array[currentIndex]>target){
      endIndex = currentIndex;
      currentIndex = (endIndex+startIndex)/2;
    }
    else{
      startIndex = currentIndex;
      currentIndex = (endIndex+startIndex)/2;
    }
    return currentIndex;
  }
  public void run(){
    while(!found && currentIndex!=updateCurrentIndex()){
       if(array[currentIndex]==target){
         System.out.println("found "+target+" at index "+currentIndex+"\nby thread : "+
Thread.currentThread().getId());
         found = true;
       }
    }
    threadsExhausted++;
  }
}
```

```
class ThreadManager{
  int lastIndex;
  int[] array;
  int target;
  ThreadManager(int[] searchArray,int target){
    lastIndex = searchArray.length;
    array = searchArray;
    Arrays.sort(array);
    this.target = target;
    for(int i = 0;i<(lastIndex+1)/10;i++){
      Thread temp = new Thread(new BinarySearch(array,i*10,i*10+9,target));
      temp.start();
    }

    while(BinarySearch.threadsExhausted!=(lastIndex+1)/10){
      try{
        Thread.currentThread().sleep(1);
      }catch(InterruptedException e){
        e.printStackTrace();
      }
    }
    if(!BinarySearch.found){
      System.out.println(target +" is not in array.");
    }
  }
}

public class Main {
    public static void main(String[] args) {
        int[] searchArray = generateRandomArray(50);
        int target = searchArray[23];
        //int target = 43;
        ThreadManager threadManager = new ThreadManager(searchArray, target);
        System.out.println("Array: " + Arrays.toString(searchArray));
    }

    private static int[] generateRandomArray(int size) {
        int[] array = new int[size];
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            array[i] = random.nextInt(100);
        }
        return array;
    }
}
```

```
[be2288@localhost 7]$ java Main
found 48 at index 25
by thread : 22
Array: [0, 0, 0, 3, 8, 10, 13, 15, 15, 16, 16, 18,
 23, 27, 29, 34, 34, 35, 38, 40, 40, 40, 41, 45, 4
6, 48, 49, 50, 50, 54, 56, 58, 60, 62, 62, 64, 67,
 70, 73, 75, 80, 87, 89, 89, 92, 93, 94, 94, 95, 9
9]
```