ASSIGNMENT-2 OOS-LAB

NAME: SAPTARSHI MAJUMDAR

IT 2nd year 2nd semester

ROLL:002211001077

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

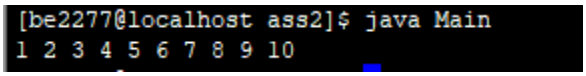
```
class Sample extends Thread {
   public void run() {
      System.out.println("In main thread..");
   }
}

class SampleChild extends Sample {
   public void run() {
      System.out.println("In child thread..");
   }
}

public class Main {
   public static void main(String[] args) {
      Sample s1 = new Sample();
      s1.start();

                SampleChild sc1 = new SampleChild();
                sc1.start();
   }
}
```

```
[be2277@localhost ass2]$ java ql.Main
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 Main {
   private static final int countTo = 10;
   private static int currentNumber = 1;
   private static final Object lock = new Object();

   public static void main(String[] args) {

      Thread evenThread = new Thread() {
         public void run() {
            synchronized (lock) {
               while(currentNumber <= countTo) {
                  System.out.print(currentNumber++ + " ");
                  lock.notify();
                  try{
                     if(currentNumber <= countTo)
                        lock.wait();
                  }
                  catch(Exception e) {}
               }
            }
         }
      };

      Thread oddThread = new Thread() {
         public void run() {
```

```
            synchronized (lock) {
                while(currentNumber <= countTo) {
                    System.out.print(currentNumber++ + " ");
                    lock.notify();
                    try{
                        if(currentNumber <= countTo)
                            lock.wait();
                    }
                    catch(Exception e) {}
                }
            }
        }
    };

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

    try {
        oddThread.join();
        evenThread.join();
    } catch (InterruptedException e) {
        System.out.println(e);
    }

    System.out.println();
  }
}
```



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.

```
class Main {
  private final static int n = 11; // series terms
  private final static Object lock = new Object();
  private static long currentDenominator = 0l;
  private static Double result = 0.0;

  private static long factorial(int n) {
    if(n == 0) return 1;
    return n * factorial(n-1);
  }

  public static void main(String[] args) {
    Thread t1 = new Thread() {
      public void run() {
        synchronized (lock) {
          for(int i=0; i<n; i++) {
            currentDenominator = factorial(i);
            lock.notify();
            try { if(i < n - 1) lock.wait(); } catch(Exception e) {}
          }
        }
      }
    };

    Thread t2 = new Thread() {
      public void run() {
```

```
            synchronized (lock) {
                for(int i=0; i<n; i++) {
                    while(currentDenominator == 0) {
                        try { lock.wait(); } catch(Exception e) {}
                    }
                    result += 1.0 / currentDenominator;
                    currentDenominator = 0;
                    lock.notify();
                }
            }
        }
    };

    t1.start();
    t2.start();

    try{
        t1.join();
        t2.join();
    }catch(Exception e) {}

    System.out.println("Result: " + result);
  }
}
```

```
[be2277@localhost ass2]$ javac q3.java
[be2277@localhost ass2]$ java Main
Result: 2.7182818011463845
```

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.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

class Main {
    public static int x = 0, y = 1;
    public static int result = 0;
    public static int currentNumber = 0;
    public static int currentNumberSign = 1;
    public static final Object lock = new Object();
    public static boolean isFileOpen;

    private static boolean isNumber(int i) {
        if(i >= 48 && i <= 57) return true;
        return false;
    }

    private static int toNumber(int i) {
        return i - 48;
    }

    public static void main(String[] args) {
        try (FileReader fin = new FileReader("./input.txt")) {
            isFileOpen = true;
            Thread producer = new Thread() {
                public void run() {
                    synchronized (lock) {
                        int i = 0;
```

```
        try {
          while((i = fin.read()) != -1) {
            currentNumber = 0;
            currentNumberSign = 1;
            if(i == 45) {
              currentNumberSign = -1;
              i = fin.read();
            }
            if(isNumber(i)) {
              currentNumber += toNumber(i);
              while((i = fin.read()) != -1 && isNumber(i)) {
                currentNumber *= 10;
                currentNumber += toNumber(i);
              }
              currentNumber *= currentNumberSign;
            }
            lock.notify();
            try {
              if(isFileOpen)
                lock.wait();
            } catch (InterruptedException e) { e.printStackTrace(); }
          }
        } catch (IOException e) {
          System.out.println("Can't read, need glasses");
        }
      }
    }
};

Thread consumer = new Thread() {
```

```
        public void run() {
            synchronized(lock) {
                while(isFileOpen) {
                    result += currentNumber;
                    lock.notify();
                    try {
                        if(fin.ready()) lock.wait();
                        else isFileOpen = false;
                    }
                    catch(InterruptedException e) { e.printStackTrace(); }
                    catch(IOException e) { System.out.println(e); }
                }
            }
        }
    };

    producer.start();
    consumer.start();


    try {
        producer.join();
        consumer.join();
    } catch (InterruptedException e) { e.printStackTrace(); }

    System.out.println("Result: " + result);
}
catch(FileNotFoundException e) {
    System.out.println("Where did the file go");
}
```

```
        catch(IOException e) {
            System.out.println(e);
        }
    }
}
```

```
[be2277@localhost ass2]$ javac q4.java
[be2277@localhost ass2]$ java Main
Result: 145
```

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 Resource {
    protected static int lastTerm;
    protected static final Object lock = new Object();
    protected static int result = 0;

    protected void printResult() {
        System.out.println("Result: " + result);
    }
}

class TOdd extends Resource implements Runnable {
    public TOdd(int lastTerm) {
        Resource.lastTerm = lastTerm;
    }

    @Override
    public void run() {
        synchronized (lock) {
```

```
            for(int i=1; i<=lastTerm; i++) {
                if(i % 2 != 0) {
                    result += i;
                }
                lock.notify();
                try {
                    if(i < lastTerm)
                        lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                if(i == lastTerm && lastTerm % 2 != 0) {
                    printResult();
                }
            }
        }
    }
}

class TEven extends Resource implements Runnable {
    public TEven(int lastTerm) {
        Resource.lastTerm = lastTerm;
    }

    @Override
    public void run() {
        synchronized (lock) {
            for(int i=1; i<=lastTerm; i++) {
                if(i % 2 == 0) {
```

```
                result += i;
            }
            lock.notify();
            try {
                if(i < lastTerm)
                    lock.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            if(i == lastTerm && lastTerm % 2 == 0) {
                printResult();
            }
        }
    }
}

class Main {
    public static void main(String[] args) {
        int lastTerm = 100;
        TOdd obj1 = new TOdd(lastTerm);
        TEven obj2 = new TEven(lastTerm);

        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);

        t1.start();
        t2.start();
    }
```

}

```
[be2277@localhost ass2]$ javac q5.java
[be2277@localhost ass2]$ java Main
Result: 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. a) Create n/10 threads t1, t2,..,tn/10. b) Distribute the numbers among threads such that ti will have numbers ai, ai+1, ….a2i-1. c) Distribute the element e to all threads. d) Each thread searches the element e in its sub-array using binary search algorithm.

```
public class Search {
   private int[] array;
   private int target;
   private int numberOfThread;
   private int threadIndexOfTarget = -1;
   private int arrayIndexOfTarget = -1;

   public Search(int[] arr, int target) {
      this.array = arr;
      this.target = target;
      numberOfThread = (array.length%10 == 0) ? array.length/10 : array.length/10 + 1;
   }

   public int getNumberOfThread() {
      return numberOfThread;
   }

   private int binarySearch(int startIndex, int endIndex) {
      while(startIndex <= endIndex) {
         int midIndex = (endIndex + startIndex) / 2;
         if(target < array[midIndex]) {
```

```
            endIndex = midIndex - 1;
        } else if(target > array[midIndex]) {
            startIndex = midIndex + 1;
        } else {
            return midIndex;
        }
    }
    return -1;
}

private int i;
public int[] searchElementUsingThreads() {
    Thread[] threads = new Thread[numberOfThread];

    for(i=0; i<numberOfThread; i++) {
        int startIndex = i * 10;
        int endIndex = Math.min(startIndex + 9, array.length - 1);
        threads[i] = new Thread(() -> {
            int index = binarySearch(startIndex, endIndex);
            if (index != -1) {
                if(threadIndexOfTarget == -1) {
                    threadIndexOfTarget = i + 1;
                    arrayIndexOfTarget = index + 1;
                }
            }
        });
        threads[i].start();
        try {
            threads[i].join();
        }
```

```
            catch (InterruptedException e) { System.out.println(e); }
        }

        return new int[]{threadIndexOfTarget, arrayIndexOfTarget}; // If not found: {-1, -1}
    }
}
```

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

```
public class Main {
    private static final int arrayLength = 100;
    private static final int target = 12;
    private static int[] answer = new int[2];

    public static void main(String[] args) {

        int[] arr = new int[arrayLength];
        for(int i=0; i<arr.length; i++) {
            arr[i] = i + 1;
        }

        Search s = new Search(arr, target);

        answer = s.searchElementUsingThreads();

        if(answer[0] == -1) {
            System.err.println("Element not found!");
        } else {
            System.out.println("Element " + target + " found at:");
```

```
        System.out.println("Thread Index: " + answer[0] + " out of " + s.getNumberOfThread() + "
Threads");

        System.out.println("Location: " + answer[1] + " element out of " + arrayLength + "
Elements");

     }

   }

}
```