OOS Lab

Name: Swapnadeep Mishra

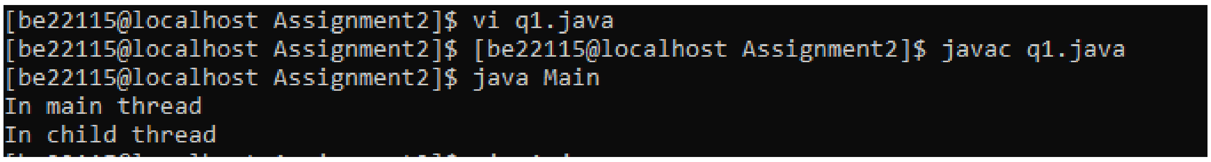
Section- A3 Roll- 002211001115

Assignment-2

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

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

public class Main {
    public static void main(String[] args) {
        System.out.println("In main thread");

        MyThread childThread = new MyThread();
        childThread.start();
    }
}
```

Output:-

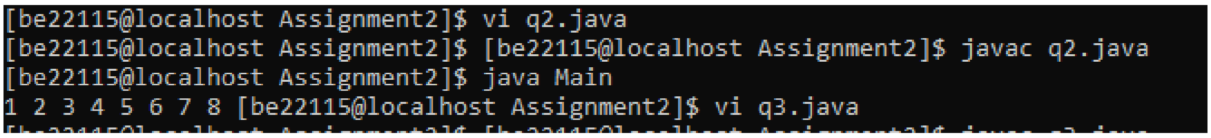


```
[be22115@localhost Assignment2]$ vi q1.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q1.java
[be22115@localhost Assignment2]$ java Main
In main thread
In child thread
```

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 EvenThread extends Thread {
    public void run() {
        synchronized (Main.lock) {
            for (int i = 2; i <= 8; i += 2) {
                System.out.print(i + " ");
                Main.lock.notify();
                try {
                    if (i < 8)
                        Main.lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

class OddThread extends Thread {
    public void run() {
        synchronized (Main.lock) {
            for (int i = 1; i <= 7; i += 2) {
                System.out.print(i + " ");
                Main.lock.notify();

                try {

                    if (i < 7)

                        Main.lock.wait();

                } catch (InterruptedException e) {

                    e.printStackTrace();

                }

            }

        }

    }

}


public class Main {

    public static final Object lock = new Object();


    public static void main(String[] args) {

        Thread evenThread = new EvenThread();

        Thread oddThread = new OddThread();


        oddThread.start();

        evenThread.start();

    }

}
```

Output:-



```
[be22115@localhost Assignment2]$ vi q2.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q2.java
[be22115@localhost Assignment2]$ java Main
1 2 3 4 5 6 7 8 [be22115@localhost Assignment2]$ vi q3.java
```

3) Consider the following series

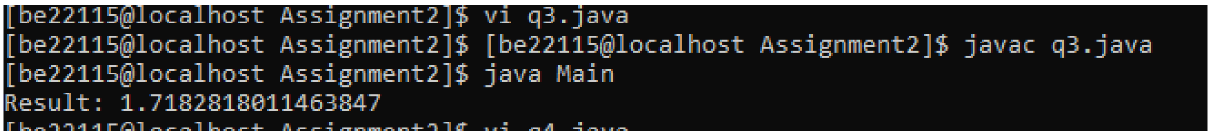
x = 1+1/1! +1/2! + 1/3! + ...1/10!

Create two threads t1 & t2. t1 will generate the denominators and t2 will form the term and add them up. Finally print the result.

```
import java.util.concurrent.*;


class DenominatorGenerator extends Thread {
    private BlockingQueue<Double> queue;


    public DenominatorGenerator(BlockingQueue<Double> queue) {
        this.queue = queue;
    }


    public void run() {
        try {
            for (int i = 1; i <= 10; i++) {
                double denominator = 1.0;
                for (int j = 1; j <= i; j++) {
                    denominator *= j;
                }
                queue.put(1.0 / denominator);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```
class TermCalculator extends Thread {
    private BlockingQueue<Double> queue;

    public TermCalculator(BlockingQueue<Double> queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            double result = 0.0;
            for (int i = 0; i < 10; i++) {
                result += queue.take();
            }
            System.out.println("Result: " + result);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BlockingQueue<Double> queue = new LinkedBlockingQueue<>();

        Thread t1 = new DenominatorGenerator(queue);
        Thread t2 = new TermCalculator(queue);
        t1.start();

        t2.start();

    }

}
```

Output:-



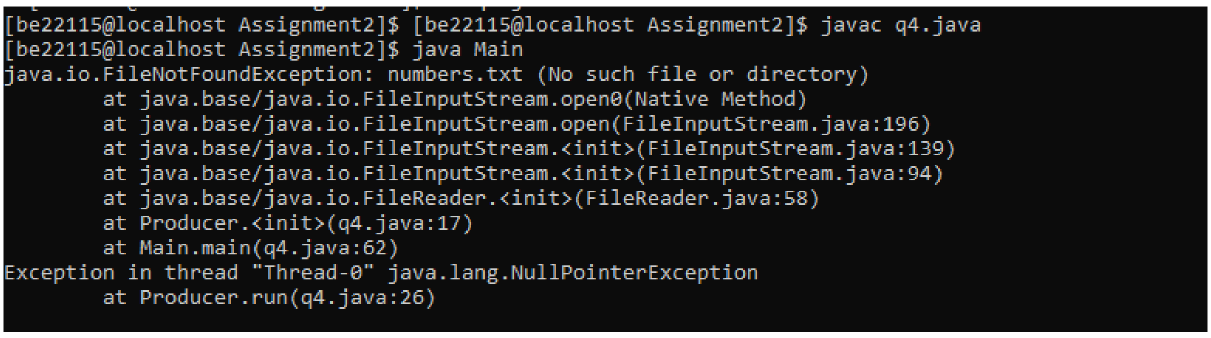
```
[be22115@localhost Assignment2]$ vi q3.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q3.java
[be22115@localhost Assignment2]$ java Main
Result: 1.7182818011463847
```

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.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;


class Producer extends Thread {


    private BlockingQueue<Integer> queue;

    private BufferedReader reader;

    private String fileName;


    public Producer(BlockingQueue<Integer> queue, String fileName) {

        this.queue = queue;
        this.fileName = fileName;
        try {
            this.reader = new BufferedReader(new FileReader(fileName));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        try {
            String line;
            while ((line = reader.readLine()) != null) {
                int number = Integer.parseInt(line.trim());
                queue.put(number);
            }
            reader.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }

    }
}

class Consumer extends Thread {
    private BlockingQueue<Integer> queue;

    public Consumer(BlockingQueue<Integer> queue) {
        this.queue = queue;
    }

    public void run() {
        int sum = 0;
        try {
            while (true) {
                int number = queue.take();
                sum += number;
                System.out.println("Consumed: " + number);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Sum: " + sum);
    }
}

public class Main {
    public static void main(String[] args) {
        BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(10);
        Producer producer = new Producer(queue, "numbers.txt");
        Consumer consumer = new Consumer(queue);

        producer.start();
        consumer.start();
    }
```

}

Output:-

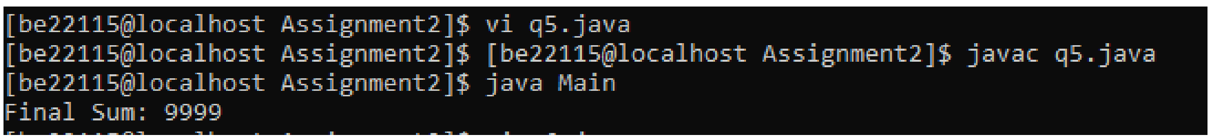


```
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q4.java
[be22115@localhost Assignment2]$ java Main
java.io.FileNotFoundException: numbers.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:196)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:139)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:94)
        at java.base/java.io.FileReader.<init>(FileReader.java:58)
        at Producer.<init>(q4.java:17)
        at Main.main(q4.java:62)
Exception in thread "Thread-0" java.lang.NullPointerException
        at Producer.run(q4.java:26)
```

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 SeriesCalculator extends Thread {

  private int start;

  private int end;

  private int result;


  public SeriesCalculator(int start, int end) {

    this.start = start;

    this.end = end;

  }


  public int getResult() {

    return result;

  }


  public void run() {

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

public class Main {
  public static void main(String[] args) {
    SeriesCalculator series1 = new SeriesCalculator(1, 99);
    SeriesCalculator series2 = new SeriesCalculator(2, 100);

    series1.start();
    series2.start();

    try {
        series1.join();
        series2.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    int finalSum = series1.getResult() + series2.getResult();
    System.out.println("Final Sum: " + finalSum);
  }
}
```

Output:-



```
[be22115@localhost Assignment2]$ vi q5.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q5.java
[be22115@localhost Assignment2]$ java Main
Final Sum: 9999
```

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.

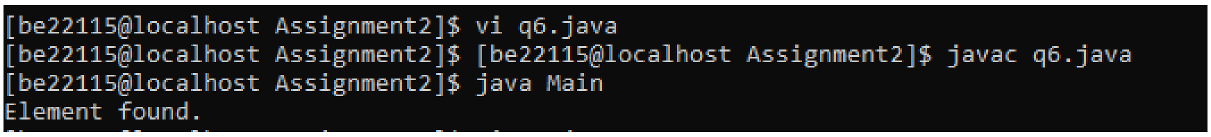
d) Each thread searches the element e in its sub-array using binary search algorithm.

```
import java.util.concurrent.*;


class ParallelBinarySearchThread extends Thread {

   private int[] array;

   private int target;

   private int startIndex;

   private int endIndex;

   private boolean found = false;


   public ParallelBinarySearchThread(int[] array, int target, int startIndex, int endIndex) {

      this.array = array;

      this.target = target;

      this.startIndex = startIndex;

      this.endIndex = endIndex;

   }


   public void run() {
        int low = startIndex;

        int high = endIndex;


        while (low <= high && !found) {

            int mid = low + (high - low) / 2;

            if (array[mid] == target) {

                found = true;

                break;

            } else if (array[mid] < target) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

    }


    public boolean isFound() {

        return found;

    }

}


public class Main {

    public static void main(String[] args) {

        int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}; // Example array

        int target = 10; // Example target element


        int numThreads = array.length / 10;
```

```
ParallelBinarySearchThread[] threads = new ParallelBinarySearchThread[numThreads];

for (int i = 0; i < numThreads; i++) {
  int startIndex = i * 10;
  int endIndex = Math.min(startIndex + 9, array.length - 1);
  threads[i] = new ParallelBinarySearchThread(array, target, startIndex, endIndex);
  threads[i].start();
}

try {
  for (ParallelBinarySearchThread thread : threads) {
    thread.join();
  }
} catch (InterruptedException e) {
  e.printStackTrace();
}

boolean found = false;
for (ParallelBinarySearchThread thread : threads) {
  if (thread.isFound()) {
    found = true;
    break;
  }
}

if (found) {
  System.out.println("Element found.");
```

```
        } else {
            System.out.println("Element not found.");
        }
    }
}
```

Output:-



```
[be22115@localhost Assignment2]$ vi q6.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q6.java
[be22115@localhost Assignment2]$ java Main
Element found.
```

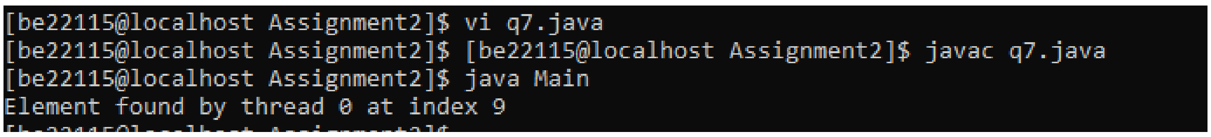
7) Write a Java program using threading technology and print the thread index and location where the element has been found.

```
class BinarySearchThread extends Thread {
    private int[] array;
    private int target;
    private int startIndex;
    private int endIndex;
    private int threadIndex;
    private int foundIndex = -1;

    public BinarySearchThread(int[] array, int target, int startIndex, int endIndex, int threadIndex) {
        this.array = array;
        this.target = target;
        this.startIndex = startIndex;
        this.endIndex = endIndex;
        this.threadIndex = threadIndex;
    }

    public void run() {
        int low = startIndex;
        int high = endIndex;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (array[mid] == target) {
                foundIndex = mid;
                break;
            } else if (array[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }

    public int getThreadIndex() {
        return threadIndex;
    }

    public int getFoundIndex() {
        return foundIndex;
    }
}

public class Main {
```

```
public static void main(String[] args) {
  int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
  int target = 10;

  int numThreads = array.length / 10;
  BinarySearchThread[] threads = new BinarySearchThread[numThreads];

  for (int i = 0; i < numThreads; i++) {
    int startIndex = i * 10;
    int endIndex = Math.min(startIndex + 9, array.length - 1);
    threads[i] = new BinarySearchThread(array, target, startIndex, endIndex, i);
    threads[i].start();
  }

  try {
    for (BinarySearchThread thread : threads) {
      thread.join();
    }
  } catch (InterruptedException e) {
    e.printStackTrace();
  }

  for (BinarySearchThread thread : threads) {
    if (thread.getFoundIndex() != -1) {
      System.out.println("Element found by thread " + thread.getThreadIndex() + " at index "
          + thread.getFoundIndex());
    }
```

```
        }

    }

}
```

Output:-



```
[be22115@localhost Assignment2]$ vi q7.java
[be22115@localhost Assignment2]$ [be22115@localhost Assignment2]$ javac q7.java
[be22115@localhost Assignment2]$ java Main
Element found by thread 0 at index 9
```