POORNIMA

Experiment - 6

6.1 Write a program to create a file and do the addition of two integers numbers and write the output on the file.

```
import java.io.* ;

public class AddToFile {
  public static void main (String [] args) {
   int n1 = 5;
   int n2 = 7;
   int sum = n1 + n2;
   File file = new File ("output.txt");
   try {
    FileWriter writer = new FileWriter (file);
    writer.write(" The sum of " + n1 + "and " + n2 + "is:"
                     + sum);
    writer.close ();
   System.out.println (" Sum has been written ~~in~~ to the
                          file successfully ");
  } catch (IOException e) {
    System.out.println (" An error occurred while
                          writing to the file ");
    e.printStackTrace ();
  }
 }
}
```

Page No......

output :-

```
The sum of 5 and 7 is : 12
Sum has been written in the file successfully
```

POORNIMA

6.2 WAP to create a file and do the addition of 2 nos. and write the o/p on the file after showing the result delete the file.

```
import java.io.*
public class Main {
public static void main (String[] args) {
 Scanner scanner = new Scanner (System.in);
 System.out.println ("Enter the first no. : ");
 double n1 = scanner.nextDouble ();
 System.out.println ("Enter the second no. : ");
double n2 = Scanner.nextDouble ();
double sum = n1 + n2;
File file = new File ("result.txt");
try {
FileWriter writer = new FileWriter (file );
Writer.write ("The sum of " + n1 + "and " + n2 + " is: " +sum);
File .delete ();
} catch ( IOException e) {
 System.out.println ("An error occurred while
                    writing to the file !");

 e. print Stack Trace ();
 }
}
}
```

Output :-

```
Enter the first number : 4
Enter the second number : 6
The sum of 4.0 and 6.0 is : 10.0
```

POORNIMA

Experiment 7

Ques: What is the difference between wait and sleep classes in java?

		Wait	Sleep
1.	Releasing locks	Releases the lock on the object it is called on, allowing other threads to acquire the lock.	Does not release any locks held by the thread.
2.	Synchronisation Context	Must be called within a Sychronised block or method	Can be called from any context, synchronised or not
3.	Purpose	Used for inter-thread Communication, to make a thread wait until some some condition is met	Used to pause the execution of the current thread for a specific duration
4.	Notification Mechanism	Requires another thread to call notify() or notifyAll() to wake up waiting thread	The thread Automatically wakes up after the specific duration
5.	Syntax	Synchronised (obj){ obj.wait(); }	Thread.sleep(long millis);

Ques: What are advantages and disadvantages of multithreading in Java?

Ans: Advantages of Multithreading in Java

1. Improproved Performance:
   - Concurrency: Better CPU Utilization, especially on multi-core processors.
   - Responsiveness: keeps GUI applications responsive by handling long tasks in the backgroud.
2. Resource Sharing:
   - Efficient Utilization: Threads share memory and other resources, which is more efficient than seprate processes.
3. Simplied Modeling:
   - Natural Structure: Easier to model real world concurrent activities
4. Asynchronous I/O:
   - Non-blocking operations: Enhance performance in I/O-intensive applications

Disadvantages of Multithreading in Java

1. Overhead:
   - Context Switching: Frequent context swithes can reduce performance
   - Resource Consumption: Managing multiple threads uses more memory and CPU.
2. Scalability Limitation:
   - Contention: High contention for shared resources can lead to bottlenecks.
3. Security Risks:
   - Inconsistent State: Improper Synchonisation can leave resources in an inconsistent state.

POORNIMA

7.3 Create a scenario where multiple threads acts as a reader, reading from a shared resource & another thread acts as a writer modifying that resource. Implement a solution that allows multiple users to access the resource simultaneously but exclusively accessed for the writer.

```
import java.util.concurrent.locks.ReentrantReadWriteLock;

class SharedResource {
        private int data;
        private final ReentrantReadWriteLock rwLOCK =
                    new ReentrantReadWriteLock();
    public void read(){
      rwLock.readLock().lock();
      try {
         System.out.println(Thread.currentThread().
         getName() + "is reading data : " + data);
         Thread.sleep(100);
     } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    } finally {
     rwLock.readLock().unlock();
   }
  }
   public void write(int newData) {
        rwLock.writeLock().lock();
   try {
```

```
System.out.println(Thread.currentThread().getName()
          + "is writing data:" + newData);
   Thread.sleep(100);
   data = newData;
  } catch (InterruptedException e) {
   Thread.currentThread().interrupt();
  } finally {
   rwLock.writeLock().unlock();
  }
} }

public class ReadWriteLockExample {
     public Static void main (String[] args) {
     SharedResource sharedResource = new SharedResource();

   Runnable readerTask = () -> {
   for (int i=0; i<4; i++) {
        SharedResource.read();
} };
   Thread writer = new Thread(writerTask, "Writer");
   reader1. Start ();
   reader2. start();
   writer. Start ();

   try {
   reader1. join();
   reader2. join();
   writer. join();
}
```

```
catch (InterruptedException e) {
    Thread.currentThread().interrupt();
  }
 }
}
```

POORNIMA

Experiment - 8

Ques: Write a program to calculate addition of two complex Numbers.

```
public class Cmx {
private double real;
private double imaginary;
public Cmx(double real, doublo imaginary) {
 this.real = real;
 this.imaginary = imaginary;
}
public Cmx add(Cmx other){
double newReal = this.real+ other.real;
double newImaginary = this.imaginary + other.imaginary;
return new Cmx (newReal, newImaginary);
}
@Override
public String toString() {
return real + "+" + imaginary + "i";
}
public static voidmain (String[]args) {
Cmx c1 = new Cmx(2.3, 4.5);
Cmx c2 = new Cmx(1.4, 3.7);
Cmx result = c1.add(c2);
System.out.println("First Complex Number: " + c1);
System.out.println("Second Complex Number: " + c2);
System.out.println("Sum of Complex Number: "+ result)
}}
```

Output:

```
First complex Number: 2.3+4.5i
Second complex Number: 1.4 + 3.7i
Sum of complex Numbers: 3.69997+8.2i
```