POORNIMA

Experiment-6

Write a program to create a file and do the addition of two integer 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 the
                             file succesfully")
    }
    catch (IOException e) {
        System.out.println ("An error occured while writing
                             to the file");
        e.printStackTrace();
    }
}
}
```

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 sc = new Scanner(System.in);
        System.out.println("Enter the first no");
        double n1 = sc.nextDouble();
        System.out.println("Enter the second no:");
        double n2 = sc.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 occured while writi
                    to the file!");
            e.printStackTrace();
        }
    }
}
```

Output,

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

output

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

Experiment - 7

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

	Wait	Sleep
Releasing locks	Release the lock on the object it is called on, allowing other threads to acquire the lock	Does not release any lock held by the thread.
Synchronization Context	Must be called within a synchronised block or method	Can be called from any context, synchronization or not
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
Notification Mechanism	Requires another thread to call notify() or notifyAll() to wake up waiting thread	The thread automatically wakes up after the specific duration

Syntax	Synchronised(obj){
	obj.wait();
	}
		Thread.sleep(long millis);

What are advantages and disadvantages of multithreading in Java?

Advantages of Multi threading in Java

Improproved Performance :

Concurrency : Better CPU utilization, especially on multicore processors.

Responsiveness : Keeps GUI applications responsive by handling long tasks in the background

Resource Sharing :

Efficient utilization : Threads share memory and other resources which is more efficient than seperate processes

Simplied Modeling :

Natural Structure : Easier to model real world concurrent activities

Disadvantages of Multithreading in Java :

Overhead :

Context Switching : Frequent context switches can reduce performance

Resource Consumption : Managing Multiple threads uses more memory and CPU

Scalability Limitations :

Contention : High contention for shared resources can lead to bottle necks

73 Create a scenario where multiple thread 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 access 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();
}
}
}
```

Output

Reader 2 is reading data : 0
Reader 1 is reading data : 0
Writer is writing data : 0
Writer is writing data : 1
Writer is writing data : 2
Writer is writing data : 3
Reader 2 is reading data : 3
Reader 1 is reading data : 3
Reader 2 is reading data : 3
Reader 1 is reading data : 3
Reader 1 is reading data : 3
Reader 2 is reading data : 3

POORNIMA

Experiment-8

Write a program to calculate addition of two complex Numbers

```
public class Cmx {
  private double real;
  private double imaginary;
  public Cmx (double real, double imaginary) {
    this.real = real;
    this.imaginary = imaginary;
  }
  public Cmx add (Cmx other) {
    double newReal = this.realt other.real;
    double newImaginary = this.imaginary + other.imagin
    return new Cmx (newReal, newImaginary);
  }
  @Override
  public String toString () {
    return real + "+" + imaginary + "i";
  }
  public static void main (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 no. :" + C1);
    System.out.println ("sec. complex no. :" + C2);
    System.out.println ("sum of complex nos :" + result);
  }
}
```

Output →

first complex no : 2.3 + 4.5i
sec. complex no : 1.4 + 3.7i
sum of complex nos : 3.69997 + 8.2i

POORNIMA

Experiment - 9

Write a program in Java to solve Tower of Hanoi Problem.

```
class Tower {
    public static void TowerOfHanoi(int n, char start, char
        end, char aux) {
    if (n == 1) {
        System.out.println("Move disc 1 from rod" + start +
                           " to rod" + end);
        return;
    }
    TowerOfHanoi(n-1, start, aux, end);
    System.out.println("Move disk" + n + " from rod" + start +
                       " to rod" + end);
    TowerOfHanoi(n-1, aux, end, start);
}

public static void main(String args[]) {
    int n = 4
    TowerOfHanoi(n, 'A', 'B', 'C');
}
}
```

21/05/24

Move disk 1 from rod A to rod B
Move disk 2 from rod B to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

Experiment-10

1. Write a java program to create a box as your (red) output
2. Write a program to change the background color on every different combination of RGB colors

(1)

```
import java.awt.*;
public class RedColorBox {
    Public RedBox() {
    Frame f = new Frame("Red Color Box");
    f.setLayout(null);
    f.setSize(400, 400);
    f.setVisible(true);
    f.setBackground(color.Red);
    f.setSize(400, 400)
  }
public static void main(String[] args) {
    new RedBox();
  }
}
```

(2)

```
import javax.swing.*;
import java.awt.*;
import java.util.Timer;
import java.util.TimerTask;
import java.awt.
public class BackgroundColorChanger extends JFrame {
private int red=0, green=0, blue=0;
```

```
public BackgroundColorChanger () {
  setTitle ("BackgroundColorChanger");
  setSize (900, 900);
  setDefaultOperation (EXIT_ON_CLOSE);
  colorPanel = new JPanel ();
  add (colorPanel, BorderLayout.CENTER);
  ChangeBackgroundColor ();
}

private void changeBackgroundColor () {
  Timer timer = new Timer (1000, e -> { if (red < 255) {
    red += 50; }
  else {
    red = 0;
    if (green < 255) {
      green += 50; }
    else {
      green = 0
      blue += 50;
    }
    else {
      blue = 0; }
  }
  timer.start ();
}

public static void main (String [] args) {
  new BackgroundColorChanger () . setVisible (true); }
}
```

24/5/24

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Polygon;

public class PentagonApplet extends Applet {

    @Override
    public void paint(Graphics g) {
        // Set the background color
        setBackground(Color.WHITE);

        // Define the coordinates of the pentagon
        int[] xPoints = {100, 140, 120, 80, 60};
        int[] yPoints = {50, 100, 150, 150, 100};
        int nPoints = 5;

        // Create the pentagon polygon
        Polygon pentagon = new Polygon(xPoints, yPoints, nPoints);

        // Set the color for the pentagon
        g.setColor(Color.BLUE);
        g.fillPolygon(pentagon);

        // Set the color for the border
        g.setColor(Color.BLACK);
        g.drawPolygon(pentagon);
    }

    @Override
    public void init() {
        // Set the size of the applet
        setSize(200, 200);
    }
}
```

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;

/*
<applet code="TriangleWithBorderApplet.class" width="400"
height="400">
</applet>
*/

public class TriangleWithBorderApplet extends Applet {

    public void paint(Graphics g) {
        // Coordinates of the triangle vertices
        int[] xPoints = { 200, 100, 300 };
        int[] yPoints = { 100, 300, 300 };
        int nPoints = 3;

        // Set the color for the triangle fill
        g.setColor(Color.YELLOW);
        g.fillPolygon(xPoints, yPoints, nPoints);

        // Set the color for the border
        g.setColor(Color.BLACK);
        g.drawPolygon(xPoints, yPoints, nPoints);
    }
}
```

Feature	`wait()`	`sleep()`	`yield()`
Class	Can be called on any object	`Thread` class method	`Thread` class method
Lock Release	Releases the lock acquired on the object	Does not release any locks	Does not release any locks
Execution Pause	Pauses the execution of the current thread	Pauses the execution of the current thread	Offers a hint to the scheduler to pause execution
Time Unit	Does not accept a specific time unit	Accepts time duration in milliseconds	Does not accept a specific time unit
Synchronization	Must be called within a synchronized context	Does not require synchronization	Does not require synchronization
Interruption	Can be interrupted by another thread	Can be interrupted by another thread	Cannot be interrupted by another thread
Wake-up Condition	Requires a call to `notify()` or `notifyAll()`	Does not require any wake-up condition	Does not require any wake-up condition
Utilization	Used for inter-thread communication and signaling	Used for introducing delay or pause in a thread	Used for cooperative thread scheduling

Different Ways of Reading a Text File in Java

Understanding how to read a file in Java is essential for a wide range of applications, from processing large datasets to parsing configuration files. Java provides several methods for this purpose, including:

1. Using Scanner class
2. Using BufferedReader class
3. Using File Reader class
4. Using Files Class

```
import java.io.*;
import java.util.*;

public class Example {

  public static void main(String[] args) throws IOException {
    String dir = "C:\\Bhavya\\Scaler\\readThisFile.txt";

    Scanner sc = new Scanner(new File(dir));
    sc.useDelimiter(" ");

    String data;
    while (sc.hasNext()) {
      data = sc.next();
      System.out.println(data);
    }

    sc.close();
  }
}
```

The Scanner class has more useful methods like next(), nextInt(), nextByte(), nextLine() etc, and we can assign any delimiter to divide the string rather than the default space. Since the Scanner class also reads the data from the stream **Line by Line**, we can use large text files to read.

Syntax:

```
Scanner sc = new Scanner(new File(PATH));
```

```
import java.io.*;
import java.util.*;

public class Example {

  public static void main(String[] args) throws IOException {
    String dir = "C:\\Bhavya\\Scaler\\readThisFile.txt";

    Scanner sc = new Scanner(new File(dir));
    sc.useDelimiter(" ");

    String data;
    while (sc.hasNext()) {
      data = sc.next();
      System.out.println(data);
    }

    sc.close();
  }
}
```

Reading a File in Java Using BufferedReader Class

When it comes to how to read a file in Java efficiently, especially large files, BufferedReader is often the go-to choice due to its buffering capability, which minimizes I/O operations.The BufferedReader class was there in Java from the start and is the fastest method to read the data, from a given file, **Line by Line**. It is used to read the data using the input stream. As the name suggests, the BufferedReader class buffers the data in the form of small packets of size 8 KB. Since it is only processing the data of 8 KB at a particular time, it is very efficient and can be used for large files.

Syntax:

```
BufferedReader br = new BufferedReader(new FileReader(PATH));
```

```java
// Java Program demonstrating file reading with FileReader class

import java.io.*;

// Main class
public class Scaler {

    // Main method
    public static void main(String[] args) throws Exception
    {

        // Specifying the file path as a parameter
        FileReader fr = new FileReader(
            "C:\\Users\\pankaj\\Desktop\\test.txt");

        // Variable declaration for loop control
        int i;

        // Loop continues until there's content to read
        while ((i = fr.read()) != -1)

            // Printing all the content of the file
            System.out.print((char)i);
    }
}
```

Reading a File in Java Using FileReader Class

FileReader is a convenient class for reading character files, making it a straightforward choice for how to rea a file in Java, especially when working with text files. This class is crafted with default assumptions on character encoding and buffer size, streamlining the file reading process.

Key Constructors:

- FileReader(File file): Initiates a new FileReader instance, with the specified File object for reading.
- FileReader(FileDescriptor fd): Constructs a FileReader object, utilizing the provided FileDescriptor for reading operations.
- FileReader(String fileName): Creates a FileReader object, taking the file name as input for reading.

```
import java.io.IOException;
import java.nio.file.FileAlreadyExistsException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FileCreator {

 public static void main(String... args) throws IOException {
  System.out.println("Using createFile() method of Files class: ");
  Path path = Paths.get("newFile.txt");
  try {
   Files.createFile(path);
   System.out.println("File Created: " + path);
  } catch (FileAlreadyExistsException e) {
   System.out.println("File already exists at Path: " + path);
  }
 }
}
```

"There are mainly three ways of creating a file through code in Java using JDK libraries:

1. Using the createFile() method of the Files class present in the java.nio package.
2. Using the createNewFile() method of the File class present in the java.io package.
3. Using the FileOutputStream(String fileName, boolean append) constructor of the FileOutputStream class present in the java.io package.

```
import java.io.*;

public class DeleteMethod {

  public static void main(String[] args) {
    File temp_file = new File(
      "C:\\Users\\User\\Downloads\\temporary_file.docm"
    ); // Object of file class
    if (temp_file.delete()) {
      System.out.println(temp_file.getName() + " is successfully
deleted");
    } else {
      System.out.println("Failed to delete " + temp_file.getName() + "
file");
    }
  }
}
```

Method 1: Using File.delete() Method

The File class in *java.io* package is used to perform various operations on files such as createNewFile(), exists(), canWrite(), canRead(), etc. To delete a file using the File class, we use its delete() method. This method doesn't take any parameters, and it returns a boolean. Let us note down a few important points regarding the delete() method in the File class.