PROGRAM 15

AIM : File Operations in Java

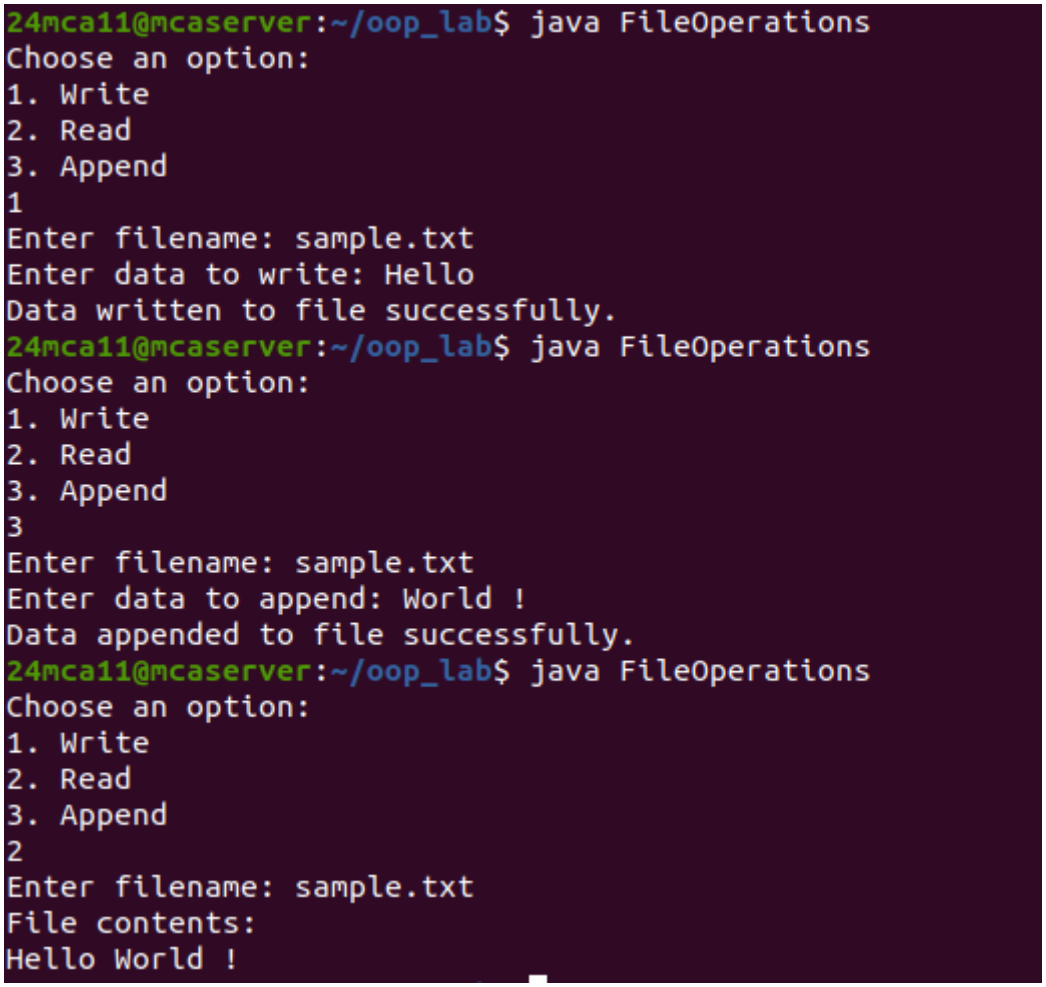
DATE : 17/03/2025

SOURCE CODE :

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

public class FileOperations {
    public static void writeFile(String filename, String data) throws IOException {
        FileWriter writer = new FileWriter(filename);
        writer.write(data);
        writer.close();
        System.out.println("Data written to file successfully.");
    }


    public static void readFile(String filename) throws IOException {
        File file = new File(filename);
        if (!file.exists()) {
            throw new FileNotFoundException("File not found.");
        }
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        String line;
        System.out.println("File contents:");
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
        reader.close();
    }


    public static void appendToFile(String filename, String data) throws IOException {
        FileWriter writer = new FileWriter(filename, true);
        writer.write(data);
        writer.close();
        System.out.println("Data appended to file successfully.");
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Choose an option:\n1. Write\n2. Read\n3. Append");
        int choice = scanner.nextInt();
        scanner.nextLine();

        System.out.print("Enter filename: ");
        String filename = scanner.nextLine();

        try {
            switch (choice) {
                case 1:
                    System.out.print("Enter data to write: ");
                    String writeData = scanner.nextLine();
                    writeFile(filename, writeData);
                    break;
                case 2:
                    readFile(filename);
                    break;
                case 3:
                    System.out.print("Enter data to append: ");
                    String appendData = scanner.nextLine();
                    appendToFile(filename, appendData);
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

OUTPUT :


24mca11@mcaserver:~/oop_lab$ java FileOperations
Choose an option:
1. Write
2. Read
3. Append
1
Enter filename: sample.txt
Enter data to write: Hello
Data written to file successfully.
24mca11@mcaserver:~/oop_lab$ java FileOperations
Choose an option:
1. Write
2. Read
3. Append
3
Enter filename: sample.txt
Enter data to append: World !
Data appended to file successfully.
24mca11@mcaserver:~/oop_lab$ java FileOperations
Choose an option:
1. Write
2. Read
3. Append
2
Enter filename: sample.txt
File contents:
Hello World !

PROGRAM 16

AIM : System-Defined and User-Defined Exception for Authentication

DATE : 17/03/2025

SOURCE CODE :

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

class AuthenticationException extends Exception {
    public AuthenticationException(String message) {
        super(message);
    }
}


public class ExceptionHandling {
    public static void readFile(String filename) throws IOException {
        File file = new File(filename);
        if (!file.exists()) {
            throw new FileNotFoundException("File not found.");
        }
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        String line;
        System.out.println("File contents:");
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
        reader.close();
    }


    public static void authenticate(String username, String password) throws
AuthenticationException {
        if (!username.equals("admin") || !password.equals("admin123")) {
            throw new AuthenticationException("Invalid username or password.");
        }
        System.out.println("Authentication successful.");
    }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter filename: ");
        String filename = scanner.nextLine();
        try {
            readFile(filename);
        } catch (FileNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO Error: " + e.getMessage());
        }

        System.out.print("Enter username: ");
        String username = scanner.nextLine();
        System.out.print("Enter password: ");
        String password = scanner.nextLine();

        try {
            authenticate(username, password);
        } catch (AuthenticationException e) {
            System.out.println("Authentication Failed: " + e.getMessage());
        }
    }
}
```

OUTPUT :

PROGRAM 17

AIM : Java Program to Perform Multithreading

DATE : 17/03/2025

SOURCE CODE :

```
import java.util.Scanner;

class SharedResource {
   boolean printMultiplication = true;
}

class MultiplicationTable extends Thread {
   private final SharedResource resource;

   public MultiplicationTable(SharedResource resource) {
      this.resource = resource;
   }

   public void run() {
      synchronized (resource) {
         for (int i = 1; i <= 10; i++) {
            while (!resource.printMultiplication) {
               try {
                  resource.wait();
               } catch (InterruptedException e) {
                  System.out.println(e.getMessage());
               }
            }
            System.out.println(i + " x 5 = " + (5 * i));
            resource.printMultiplication = false;
            resource.notify();
            try {
               Thread.sleep(500);
            } catch (InterruptedException e) {
               System.out.println(e.getMessage());
            }
         }
      }
   }
}

class PrimeNumbers extends Thread {
```

```
    private final SharedResource resource;
    private int N;

    public PrimeNumbers(SharedResource resource, int N) {
        this.resource = resource;
        this.N = N;
    }

    public void run() {
        synchronized (resource) {
            int count = 0, num = 2;
            while (count < N) {
                while (resource.printMultiplication) {
                    try {
                        resource.wait();
                    } catch (InterruptedException e) {
                        System.out.println(e.getMessage());
                    }
                }
                if (isPrime(num)) {
                    System.out.println("Prime: " + num);
                    count++;
                    resource.printMultiplication = true;
                    resource.notify();
                    try {
                        Thread.sleep(300);
                    } catch (InterruptedException e) {
                        System.out.println(e.getMessage());
                    }
                }
                num++;
            }
        }
    }

    private boolean isPrime(int num) {
        if (num < 2) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }
}

public class MultithreadingExample {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of prime numbers to generate: ");
        int N = scanner.nextInt();

        SharedResource resource = new SharedResource();

        MultiplicationTable tableThread = new MultiplicationTable(resource);
        PrimeNumbers primeThread = new PrimeNumbers(resource, N);

        tableThread.start();
        primeThread.start();

        // Wait for both threads to finish
        try {
            tableThread.join();
            primeThread.join();
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }

        System.out.println("Multithreading demonstration completed.");
        scanner.close();
    }
}
```

OUTPUT :

```
24mca11@mcaserver:~/oop_lab$ java MultithreadingExample
Enter the number of prime numbers to generate: 10
1 x 5 = 5
Prime: 2
2 x 5 = 10
Prime: 3
3 x 5 = 15
Prime: 5
4 x 5 = 20
Prime: 7
5 x 5 = 25
Prime: 11
6 x 5 = 30
Prime: 13
7 x 5 = 35
Prime: 17
8 x 5 = 40
Prime: 19
9 x 5 = 45
Prime: 23
10 x 5 = 50
Prime: 29
Multithreading demonstration completed.
```