**TERM WORK 1**

/\*1]Write a menu driven Java program to read contents of a file and :

a) print characters on the console – one character at a time

b) print the entire file c) print contents to another file.

Read both source & target file names & check for their existence/ non – existence to take appropriate actions.

\*/

import java.io.\*;

import java.util.Scanner;

public class FileOperations {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the name of the source file: ");

String sourceFileName = scanner.nextLine();

System.out.println("Enter the name of the target file (for option c): ");

String targetFileName = scanner.nextLine();

// Check if the source file exists

File sourceFile = new File(sourceFileName);

if (!sourceFile.exists()) {

System.out.println("Source file does not exist.");

return;

}

// Option menu

System.out.println("Choose an operation:");

System.out.println("a) Print characters on the console - one character at a time");

System.out.println("b) Print the entire file");

System.out.println("c) Print contents to another file");

char choice = scanner.nextLine().charAt(0);

switch (choice) {

case 'a':printCharactersOneByOne(sourceFile);

break;

case 'b':printEntireFile(sourceFile);

break;

case 'c':copyFileContents(sourceFile, targetFileName);

break;

default:System.out.println("Invalid choice");

}

}

private static void printCharactersOneByOne(File file) {

try {

BufferedReader reader = new BufferedReader(new FileReader(file));

int c;

while ((c = reader.read()) != -1) {

System.out.print((char) c);

}

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void printEntireFile(File file) {

try {

BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void copyFileContents(File sourceFile, String targetFileName) {

try {

File targetFile = new File(targetFileName);

if (targetFile.exists()) {

System.out.println("Target file already exists. Overwriting...");

}

BufferedReader reader = new BufferedReader(new FileReader(sourceFile);

BufferedWriter writer = new BufferedWriter(new FileWriter(targetFile));

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine();

}

reader.close();

writer.close();

System.out.println("Contents copied to the target file.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**TERM WORK 2**

/\*Write a Java Program to demonstrate the implementation of stream classes in Java. Assume that an input file named "input.txt" already exists with few lines of random text. Accept a filename from the user, this will be the destination file.

Write a menu driven program to do the following:

1) Transfer the contents of the input file to the destination file using the ByteArrayInputStream /ByteArrayInputStream class

3) Display the contents of the destination file.\*/

import java.io.\*;

public class StreamDemo {

public static void main(String[] args) {

try {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String inputFileName = "input.txt"; // Assume the input file is named "input.txt"

System.out.println("Enter the destination file name: ");

String outputFileName = br.readLine();

// Menu-driven options

System.out.println("Choose an operation:");

System.out.println("1) Transfer the contents of the input file to the destination file");

System.out.println("2) Display the contents of the destination file");

int choice = Integer.parseInt(br.readLine());

switch (choice) {

case 1:

transferContents(inputFileName, outputFileName);

System.out.println("Contents transferred successfully.");

break;

case 2:

displayContents(outputFileName);

break;

default:

System.out.println("Invalid choice.");

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void transferContents(String inputFileName, String outputFileName) {

try(FileInputStream input = new FileInputStream(inputFileName);

FileOutputStream output = new FileOutputStream(outputFileName)){

byte[] buffer = new byte[1024];

int byteRead;

while ((bytesRead = input.read(buffer)) != -1){

output.write(buffer,0,byteRead);

}

}catch(IOException e){

e.printStackTrace();

}

}

private static void displayContents(String outputFileName) {

try(FileInputStream input = new FileInputStream(inputFileName);

ByteArrayOutputStream output = new ByteArrayOutputStream()){

byte[] buffer = new byte[1024];

int byteRead;

while ((bytesRead = input.read(buffer)) != -1){

output.write(buffer,0,byteRead);

}

System.out.println("Contents of the destination file");

System.out.println(output.toString());

}catch(IOException e){

e.printStackTrace();

}

}

**TERM WORK 4**

import java.util.ArrayList;

import java.util.Scanner;

public class Main {

private ArrayList<Integer> integerList;

private ArrayList<Float> floatList;

public Main() {

integerList = new ArrayList<>();

floatList = new ArrayList<>();

}

// Method to add elements to the integer ArrayList

public void addElement(int element) {

integerList.add(element);

}

// Method to add elements to the float ArrayList

public void addElement(float element) {

floatList.add(element);

}

// Method to remove elements from the integer ArrayList

public void removeElement(int index) {

if (index >= 0 && index < integerList.size()) {

integerList.remove(index);

} else {

System.out.println("Invalid index for integer ArrayList!");

}

}

// Method to remove elements from the float ArrayList

public void removeElement(float element) {

if (floatList.contains(element)) {

floatList.remove(element);

} else {

System.out.println("Element not found in float ArrayList!");

}

}

// Method to perform linear search on the integer ArrayList

public boolean linearSearch(int key) {

return integerList.contains(key);

}

// Method to perform linear search on the float ArrayList

public boolean linearSearch(float key) {

return floatList.contains(key);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Main operations = new Main();

while (true) {

System.out.println("Menu:");

System.out.println("1. Add an integer element");

System.out.println("2. Add a float element");

System.out.println("3. Remove an integer element");

System.out.println("4. Remove a float element");

System.out.println("5. Search in integer ArrayList");

System.out.println("6. Search in float ArrayList");

System.out.println("7. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter an integer element: ");

int intElement = scanner.nextInt();

operations.addElement(intElement);

break;

case 2:

System.out.print("Enter a float element: ");

float floatElement = scanner.nextFloat();

operations.addElement(floatElement);

break;

case 3:

System.out.print("Enter the index of the integer element to remove: ");

int intIndex = scanner.nextInt();

operations.removeElement(intIndex);

break;

case 4:

System.out.print("Enter the float element to remove: ");

float floatElementToRemove = scanner.nextFloat();

operations.removeElement(floatElementToRemove);

break;

case 5:

System.out.print("Enter the integer to search in the list: ");

int intSearchKey = scanner.nextInt();

boolean intFound = operations.linearSearch(intSearchKey);

System.out.println("Integer found: " + intFound);

break;

case 6:

System.out.print("Enter the float to search in the list: ");

float floatSearchKey = scanner.nextFloat();

boolean floatFound = operations.linearSearch(floatSearchKey);

System.out.println("Float found: " + floatFound);

break;

case 7:

System.out.println("Exiting program...");

System.exit(0);

break;

default:

System.out.println("Invalid choice! Please enter a valid option.");

}

}

}

}

**TERM WORK 3**

import java.io.\*;

public class BinaryReadWriteDemo {

public static void main(String[] args) {

try {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter the source file name: ");

String sourceFileName = reader.readLine();

System.out.print("Enter the destination file name: ");

String destinationFileName = reader.readLine();

System.out.print("Enter text to be written to the source file: ");

String text = reader.readLine();

writeTextToFile(sourceFileName, text);

copyAlternateBytes(sourceFileName, destinationFileName);

compareFileProperties(sourceFileName, destinationFileName);

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void writeTextToFile(String fileName, String text) throws IOException {

try (FileOutputStream fos = new FileOutputStream(fileName)) {

fos.write(text.getBytes());

}

}

private static void copyAlternateBytes(String sourceFile, String destFile) throws IOException {

try (FileInputStream fis = new FileInputStream(sourceFile);

FileOutputStream fos = new FileOutputStream(destFile)) {

int data;

int count = 0;

while ((data = fis.read()) != -1) {

if (count % 2 == 0) {

fos.write(data);

}

count++;

}

}

}

private static void compareFileProperties(String file1, String file2) {

File sourceFile = new File(file1);

File destFile = new File(file2);

if (sourceFile.exists() && destFile.exists()) {

System.out.println("File Properties Comparison:");

System.out.println("Source File Name: " + sourceFile.getName());

System.out.println("Destination File Name: " + destFile.getName());

System.out.println("Source File Length: " + sourceFile.length() + " bytes");

System.out.println("Destination File Length: " + destFile.length() + " bytes");

System.out.println("Source File Readable: " + sourceFile.canRead());

System.out.println("Destination File Writable: " + destFile.canWrite());

} else {

System.out.println("One or both files do not exist.");

}

}

}

**TERM WORK 6**

class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

// Synchronized method to deposit money into the account

public synchronized void deposit(double amount) {

balance += amount;

System.out.println("Deposited: $" + amount);

// Notify a single waiting thread that a deposit has been made

notify();

}

// Synchronized method to withdraw money from the account

public synchronized void withdraw(double amount) throws InterruptedException {

while (balance < amount) {

System.out.println("Insufficient balance. Waiting for deposit...");

// Wait until sufficient funds are available

wait();

}

balance -= amount;

System.out.println("Withdrawn: $" + amount);

// Notify all waiting threads that withdrawal is completed

notifyAll();

}

// Synchronized method to get the current balance

public synchronized double getBalance() {

return balance;

}

}

public class BankDemo {

public static void main(String[] args) {

// Create a bank account with an initial balance of $1000

BankAccount account = new BankAccount(1000);

// Create a withdraw thread

Thread withdrawThread = new Thread(() -> {

try {

account.withdraw(700);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

// Create a deposit thread

Thread depositThread = new Thread(() -> {

account.deposit(500);

});

// Start the withdraw and deposit threads

withdrawThread.start();

depositThread.start();

// Wait for both threads to complete

try {

withdrawThread.join();

depositThread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

// Display the final balance after both transactions

System.out.println("Final balance: $" + account.getBalance());

}

}

**TERM WORK 7**

import java.util.LinkedList;

class Buffer {

private LinkedList<Integer> items = new LinkedList<>();

private int capacity;

public Buffer(int capacity) {

this.capacity = capacity;

}

public synchronized void produce(int item, String producerName) throws InterruptedException {

while (items.size() == capacity) {

System.out.println(producerName + " is waiting. Buffer is full.");

wait();

}

items.add(item);

System.out.println(producerName + " produced: " + item);

notifyAll();

}

public synchronized int consume(String consumerName) throws InterruptedException {

while (items.isEmpty()) {

System.out.println(consumerName + " is waiting. Buffer is empty.");

wait();

}

int consumedItem = items.removeFirst();

System.out.println(consumerName + " consumed: " + consumedItem);

notifyAll();

return consumedItem;

}

}

class Producer implements Runnable {

private Buffer buffer;

private String name;

public Producer(Buffer buffer, String name) {

this.buffer = buffer;

this.name = name;

}

@Override

public void run() {

try {

for (int i = 1; i <= 5; i++) {

buffer.produce(i, name);

Thread.sleep(1000); // Simulate production time

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

class Consumer implements Runnable {

private Buffer buffer;

private String name;

public Consumer(Buffer buffer, String name) {

this.buffer = buffer;

this.name = name;

}

@Override

public void run() {

try {

for (int i = 0; i < 5; i++) {

buffer.consume(name);

Thread.sleep(1500); // Simulate consumption time

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class Main {

public static void main(String[] args) {

Buffer buffer = new Buffer(3);

Thread producerThread = new Thread(new Producer(buffer, "Producer"));

Thread consumerThread = new Thread(new Consumer(buffer, "Consumer"));

producerThread.start();

consumerThread.start();

}

}

**TERM WORK 8**

package twjdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class TWJDBC {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/empdb";

String username = "root";

String password = "root";

try (Connection connection = DriverManager.getConnection(url, username, password)) {

String sql = "Select employee\_id, first\_name FROM employee";

try (PreparedStatement statement = connection.prepareStatement(sql)) {

try (ResultSet resultSet = statement.executeQuery()) {

while (resultSet.next()) {

int employeeId = resultSet.getInt("employee\_id");

String firstName = resultSet.getString("first\_name");

System.out.println("Employee ID: " + employeeId + ", First Name: " + firstName);

}

}

}

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**TERM WORK 9**

package twtransactionproc;

import java.sql.\*;

public class TWTransactionProc {

public static void main(String[] args) {

String jdbcUrl = "jdbc:mysql://localhost:3306/testdb";

String username = "root";

String password = "root";

try (Connection connection = DriverManager.getConnection(jdbcUrl, username, password)) {

connection.setAutoCommit(false);

try {

Savepoint savepoint = connection.setSavepoint();

insertTransaction(connection, 1, 100);

insertTransaction(connection, 2, 200);

connection.commit();

System.out.println("Transaction commit successfully.");

Savepoint savepoint2 = connection.setSavepoint();

insertTransaction(connection, 3, 300);

insertTransaction(connection, 4, 400);

connection.rollback(savepoint2);

System.out.println("Rollback to Savepoint 2.");

connection.commit();

System.out.println("Transaction committed after rollback.");

} catch (SQLException e) {

connection.rollback();

System.err.println("Transaction rolled back due to an exception.");

e.printStackTrace();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

private static void insertTransaction(Connection connection, int id, int amount) throws SQLException {

String insertQuery = "INSERT INTO transactions (id, amount) VALUES (?, ?)";

try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {

preparedStatement.setInt(1, id);

preparedStatement.setInt(2, amount);

preparedStatement.executeUpdate();

System.out.println("Inserted transaction with id=" + id + " and amount=" + amount);

}

}

}

**TERM WORK 10**