package Day10;

import java.io.\*;

import java.util.\*;

import java.nio.file.\*;

import java.nio.charset.StandardCharsets;

import java.io.Serializable;

public class FileTasks {

// Q1: Create new text file named test.txt

public static void createFile() {

File file = new File("test.txt");

try {

if (file.createNewFile()) System.out.println("Q1: test.txt created.");

else System.out.println("Q1: test.txt already exists.");

} catch (IOException e) { e.printStackTrace(); }

}

// Q2: Check whether file exists at given path

public static void checkFileExists() {

File file = new File("test.txt");

System.out.println("Q2: File exists? " + file.exists());

}

// Q3: Write "Hello, World!" to a file

public static void writeHelloWorld() {

try (FileWriter fw = new FileWriter("test.txt")) {

fw.write("Hello, World!\n");

System.out.println("Q3: Written 'Hello, World!' to test.txt");

} catch (IOException e) { e.printStackTrace(); }

}

// Q4: Read content line by line using BufferedReader

public static void readFile() {

System.out.println("Q4: Reading test.txt:");

try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

String line;

while ((line = br.readLine()) != null) System.out.println(line);

} catch (IOException e) { e.printStackTrace(); }

}

// Q5: Append a line of text to an existing file

public static void appendLine() {

try (FileWriter fw = new FileWriter("test.txt", true)) {

fw.write("This is an appended line.\n");

System.out.println("Q5: Line appended to test.txt");

} catch (IOException e) { e.printStackTrace(); }

}

// Q6: Count lines, words, characters in a file

public static void countLinesWordsChars() {

int lines = 0, words = 0, chars = 0;

try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

String line;

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

lines++;

words += line.split("\\s+").length;

chars += line.length();

}

System.out.println("Q6: Lines=" + lines + ", Words=" + words + ", Characters=" + chars);

} catch (IOException e) { e.printStackTrace(); }

}

// Q7: Copy content from one file to another

public static void copyFile() {

try (FileReader fr = new FileReader("test.txt");

FileWriter fw = new FileWriter("copy.txt")) {

int ch;

while ((ch = fr.read()) != -1) fw.write(ch);

System.out.println("Q7: Content copied to copy.txt");

} catch (IOException e) { e.printStackTrace(); }

}

// Q8: List all files in a directory

public static void listFilesInDirectory() {

File dir = new File(".");

System.out.println("Q8: Files in current directory:");

for (File f : dir.listFiles()) if (f.isFile()) System.out.println(f.getName());

}

// Q9: Filter and display only .txt files using FilenameFilter

public static void listTxtFiles() {

File dir = new File(".");

FilenameFilter filter = (d, name) -> name.endsWith(".txt");

String[] files = dir.list(filter);

System.out.println("Q9: .txt files:");

for (String f : files) System.out.println(f);

}

// Q10: Serialize and deserialize a Student object

static class Student implements Serializable {

int id; String name; int marks;

Student(int id, String name, int marks) { this.id = id; this.name = name; this.marks = marks; }

public String toString() { return "ID=" + id + ", Name=" + name + ", Marks=" + marks; }

}

public static void serializeStudent() {

Student s = new Student(1, "Alice", 95);

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"))) {

oos.writeObject(s);

System.out.println("Q10: Student serialized to student.ser");

} catch (IOException e) { e.printStackTrace(); }

}

public static void deserializeStudent() {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {

Student s = (Student) ois.readObject();

System.out.println("Q10: Deserialized Student: " + s);

} catch (IOException | ClassNotFoundException e) { e.printStackTrace(); }

}

// Q11: Read file using Scanner and display tokens

public static void readFileUsingScanner() {

try (Scanner sc = new Scanner(new File("test.txt"))) {

System.out.println("Q11: Tokens in test.txt:");

while (sc.hasNext()) System.out.println(sc.next());

} catch (FileNotFoundException e) { e.printStackTrace(); }

}

// Q12: Search for specific word and count occurrences

public static void countWordOccurrences() {

String search = "Hello";

int count = 0;

try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

String line;

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

String[] tokens = line.split("\\s+");

for (String token : tokens) if (token.equals(search)) count++;

}

System.out.println("Q12: '" + search + "' occurs " + count + " times.");

} catch (IOException e) { e.printStackTrace(); }

}

// Q13: Create, move, and delete file using Files and Paths

public static void createMoveDeleteFile() {

try {

Path file = Paths.get("tempfile.txt");

Files.createFile(file);

System.out.println("Q13: tempfile.txt created.");

Path moved = Paths.get("tempdir/tempfile.txt");

Files.createDirectories(moved.getParent());

Files.move(file, moved, StandardCopyOption.REPLACE\_EXISTING);

System.out.println("Q13: File moved to tempdir/tempfile.txt");

Files.delete(moved);

System.out.println("Q13: File deleted.");

} catch (IOException e) { e.printStackTrace(); }

}

// Q14: Read all lines using Files.readAllLines()

public static void readAllLines() {

try {

List<String> lines = Files.readAllLines(Paths.get("test.txt"), StandardCharsets.UTF\_8);

System.out.println("Q14: Lines in test.txt:");

lines.forEach(System.out::println);

} catch (IOException e) { e.printStackTrace(); }

}

// Q15: Write data and append using Files.write()

public static void writeAndAppend() {

try {

List<String> lines = Arrays.asList("Line1", "Line2");

Files.write(Paths.get("file15.txt"), lines, StandardCharsets.UTF\_8);

Files.write(Paths.get("file15.txt"), Arrays.asList("Appended Line"), StandardCharsets.UTF\_8, StandardOpenOption.APPEND);

System.out.println("Q15: Data written and appended to file15.txt");

} catch (IOException e) { e.printStackTrace(); }

}

// Q16: Walk through directory tree using Files.walk()

public static void walkDirectoryTree() {

try {

System.out.println("Q16: Files in directory tree:");

Files.walk(Paths.get(".")).filter(Files::isRegularFile).forEach(System.out::println);

} catch (IOException e) { e.printStackTrace(); }

}

// Q17: Copy file using Files.copy()

public static void copyFileWithFiles() {

try {

Files.copy(Paths.get("test.txt"), Paths.get("file17.txt"), StandardCopyOption.REPLACE\_EXISTING);

System.out.println("Q17: test.txt copied to file17.txt");

} catch (IOException e) { e.printStackTrace(); }

}

// Q18: Check and print size of file using Files.size()

public static void checkFileSize() {

try {

long size = Files.size(Paths.get("test.txt"));

System.out.println("Q18: Size of test.txt = " + size + " bytes");

} catch (IOException e) { e.printStackTrace(); }

}

// Q19: Serialize Employee object

static class Employee implements Serializable {

int id; String name; double salary;

Employee(int id, String name, double salary) { this.id = id; this.name = name; this.salary = salary; }

public String toString() { return "ID=" + id + ", Name=" + name + ", Salary=" + salary; }

}

public static void serializeEmployee() {

Employee e = new Employee(101, "Bob", 50000);

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("employee.ser"))) {

oos.writeObject(e);

System.out.println("Q19: Employee serialized to employee.ser");

} catch (IOException ex) { ex.printStackTrace(); }

}

// Q20: Deserialize

public static void main(String[] args) {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("employee.ser"))) {

Employee emp = (Employee) ois.readObject();

System.out.println("Deserialized Employee:");

System.out.println(emp);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}