=============== **File handling** ==============

1. **Write JAVA programs to write data into a file.**

**Case 1: Writing some value into a file.**

*// Practice the following piece of code*

/\* Java FileOutputStream : Write a byte into a file \*/

import java.io.FileOutputStream;

public class Demonstration\_122a {

public static void main(String args[]){

try{

FileOutputStream fout=new FileOutputStream("D:/test.txt");

fout.write(65);

fout.close();

System.out.println("File writing is over...");

}

catch(Exception e){

System.out.println(e);

}

}

}

**Case 2: Writing an array of elements into a file.**

*// Practice the following piece of code*

/\* Java FileOutputStream : Writing an array of elements into a file \*/

import java.io.\*;

class Demonstration\_122b {

public static void main(String args[]) {

byte cities[]={'D','E','L','H','I',' ','M','A','D','R','A','S','\n'};

FileOutputStream outfile=null; //create an output file stream

try {

outfile = new FileOutputStream("D:/city.txt");

// Connect the outfile stream to "city.txt"

outfile.write(cities); //Write data to the stream

outfile.close();

}

catch(IOException ioe) {

System.out.println(ioe);

System.exit(-1);

}

}

}

**Case 3: Writing a string into a file.**

*// Practice the following piece of code*

/\* Java FileOutputStream Write a string into a file \*/

import java.io.FileOutputStream;

public class Demonstration\_123 {

public static void main(String args[]){

try{

FileOutputStream fout=new FileOutputStream("D:/text.txt");

String s="Welcome to Software Engineering Lab!";

byte b[]=s.getBytes();

//converting string into byte array

fout.write(b);

fout.close();

System.out.println("File writing is over...");

}catch(Exception e){

System.out.println(e);

}

}

}

**Case 4: Read anything from the keyboard and then write the same into a file.**

*Write the code yourself….*

**2. Write JAVA programs to read from a file and print the content on the screen.**

**Case 1: Reading one character from a file.**

*// Practice the following piece of code*

/\* Java FileInputStream : Reading a single character from a file \*/

import java.io.FileInputStream;

public class Demonstration\_124 {

public static void main(String args[]){

try{

FileInputStream fin=new FileInputStream("D:\text.txt");

//Assume that text.txt is already exist.

int i=fin.read();

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

fin.close();

}catch(Exception e){System.out.println(e);}

}

}

**Case 2: Reading text from a file.**

*// Practice the following piece of code*

/\* Java FileInputStream : Reading a text from a file \*/

import java.io.FileInputStream;

public class Demonstration\_125 {

public static void main(String args[]){

try{

FileInputStream fin=new FileInputStream("D:/text.txt");

//Assume that text.txt is already exist.

int i=0;

while((i=fin.read())!=-1){

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

}

fin.close();

}catch(Exception e){System.out.println(e);}

}

}

**3. Write a JAVA program to read and make a copy of a file.**

**Case 1: Using CharacterStream class**

*// Practice the following piece of code*

/\* Copying a file into another file using CharacterStream Class \*/

import java.io.\*;

class Demonstration\_127 {

public static void main (String args[]){

//Declare and create input and output files

File inFile = new File("D:/input.dat");

File outFile = new File("D:/output.dat");

FileReader ins = null; // Creates file stream ins

FileWriter outs = null; // Creates file stream outs

try {

ins = new FileReader (inFile);

// Opens inFile

outs = new FileWriter (outFile);

// Opens outFile

int ch; // Read and write till the end

while ((ch = ins.read()) != -1){

outs.write(ch) ;

}

} catch(IOException e) {

System.out.println(e);

System.exit(-1);

} finally{ //Close files

try {

ins.close();

outs.close();

}

catch (IOException e) { }

}

} // main

} // class

**Case 2: Using ByteStream class**

*// Practice the following piece of code*

/\* Copying a file into another file using ByteStream Class \*/

import java.io.\*;

class Demonstration\_128{

public static void main (String args[]){

//Declare input and output file streams

FileInputStream infile = null ; //Input stream

FileOutputStream outfile = null ; //Output stream

byte byteRead; //Declare a variable to hold a byte

try {

//Connect infile to in.dat

infile = new FileInputStream("D:/input.dat");

//Connect outfile to out.dat

outfile = new FileOutputStream("D:/out.dat");

//Reading bytes from in.dat and writing to out.dat

byteRead = (byte)infile.read();

while(byteRead != - 1){

outfile.write(byteRead);

byteRead = (byte)infile.read();

}

}

catch(FileNotFoundException e) {

System.out.println("File not found");

}

catch(IOException e) {

System.out.println(e.getMessage());

}

finally{ //Close files

try {

infile.close();

outfile.close();

}

catch(IOException e){}

}

}

}

**4. Write a JAVA program to merge two files into a single file.**

*// Practice the following piece of code*

import java.io.\*;

class Demonstration\_1212{

public static void main (String args[]) throws IOException {

//Declare file streams

FileInputStream file1 = null;

FileInputStream file2 = null;

SequenceInputStream file3 = null; //Declare file3 to store combined files

//Open the files to be concatenated

file1 = new FileInputStream("D:/input1.txt");

file2 = new FileInputStream("D:/input2.txt");

file3 = new SequenceInputStream(file1,file2) ; //Concatenate filel and file2

//Create buffered input and output streams

BufferedInputStream inBuffer = new BufferedInputStream(file3);

BufferedOutputStream outBuffer = new BufferedOutputStream(System.out);

//Read and wri te till the end of buffers

int ch;

while((ch = inBuffer.read()) != -1)

outBuffer.write((char)ch);

inBuffer.close();

outBuffer.close();

file1.close();

file2.close();

}

}

5. Write a JAVA program to do the following.

a) Define a class **Student.** Assume suitable date and methods in it.

b) Create 10 objects of class Student and store them in an array.

c) Store all the objects into a file using **random access file**. A practice program is given for you

ready reference how to use random access file.

d) Write **another program** which will open the above created file and store them in an array.

e) Print the objects on the screen.

**Example for handling random access file.**

import java.io.\*;

class Demonstration\_1213{

public static void main (String args[])

{

RandomAccessFile file = null;

try {

file = new RandomAccessFile("rand.dat","rw");

// Writing to the file

file.writeChar('X');

file.writeInt(555);

file.writeDouble(3.1412);

file.seek (0); // Go to the beginning

// Reading from the file

System.out.println(file.readChar());

System.out.println(file.readInt());

System.out.println(file.readDouble());

file.seek(2); // Go to the second item

System.out.println(file.readInt());

// Go to the end and append false to the file

file.seek(file.length());

file.writeBoolean(false);

file.seek(4) ;

System.out.println(file.readBoolean());

file.close();

}

catch(IOException e)

{

System.out.println(e);

}

}

}

=============== **Multithreading programming**==============

1. **Write a multi-threading program using class Thread.**

*// Practice the following piece of code*

/\* Creating three threads using the class Thread and then running them concurrently. \*/

class ThreadA extends Thread{

public void run( ) {

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

System.out.println("From Thread A with i = "+ -1\*i);

}

System.out.println("Exiting from Thread A ...");

}

}

class ThreadB extends Thread {

public void run( ) {

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

System.out.println("From Thread B with j= "+2\* j);

}

System.out.println("Exiting from Thread B ...");

}

}

class ThreadC extends Thread{

public void run( ) {

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

System.out.println("From Thread C with k = "+ (2\*k-1));

}

System.out.println("Exiting from Thread C ...");

}

}

public class Demonstration\_111 {

public static void main(String args[]) {

ThreadA a = new ThreadA();

ThreadB b = new ThreadB();

ThreadC c = new ThreadC();

a.start();

b.start();

c.start();

System.out.println("... Multithreading is over ");

}

}

1. **Repeat the same program with Runnable interface.**

*// Practice the following piece of code*

/\* Creating three threads using the Runnable interface and then running them concurrently. \*/

class ThreadX implements Runnable{

public void run( ) {

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

System.out.println("Thread X with i = "+ -1\*i);

}

System.out.println("Exiting Thread X ...");

}

}

class ThreadY implements Runnable {

public void run( ) {

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

System.out.println("Thread Y with j = "+ 2\*j);

}

System.out.println("Exiting Thread Y ...");

}

}

class ThreadZ implements Runnable{

public void run( ) {

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

System.out.println("Thread Z with k = "+ (2\*k-1));

}

System.out.println("Exiting Thread Z ...");

}

}

public class Demonstration\_112 {

public static void main(String args[]) {

ThreadX x = new ThreadX();

Thread t1 = new Thread(x);

ThreadY y = new ThreadY();

Thread t2 = new Thread(y);

//ThreadZ z = new ThreadZ();

//Thread t3 = new Thread(z);

Thread t3 = new Thread(new ThreadZ());

t1.start();

t2.start();

t3.start();

System.out.println("... Multithreading is over ");

}

}

1. **Create an array of 50 numbers (numbers should be generated randomly). Write three threads to sort the numbers using three different sorting algorithms, namely Bubble sort, Quick sort and Merge sort. Once a sorted array is available, run a code for searching any element in it using Binary search algorithm.**
2. **Write a Java program for the execution of multiple transactions with an account concurrently.**

*// Practice the following piece of code and understand its working. You are advised to run the program with many trials so that you can find an inconsistency in the transactions.*

class Account {

public int balance;

public int accountNo;

void displayBalance() {

System.out.println("Account No:" +accountNo+ "Balance: " + balance);

}

void deposit(int amount){

balance = balance + amount;

System.out.println( amount + " is deposited");

displayBalance();

}

void withdraw(int amount){

balance = balance - amount;

System.out.println( amount + " is withdrawn");

displayBalance();

}

}

class TransactionDeposit implements Runnable{

int amount;

Account accountX;

TransactionDeposit(Account x, int amount){

accountX = x;

this.amount = amount;

new Thread(this).start();

}

public void run(){

accountX.deposit(amount);

}

}

class TransactionWithdraw implements Runnable{

int amount;

Account accountY;

TransactionWithdraw(Account y, int amount) {

accountY = y;

this.amount = amount;

new Thread(this).start();

}

public void run(){

accountY.withdraw(amount);

}

}

class Demonstration\_119{

public static void main(String args[]) {

Account ABC = new Account();

ABC.balance = 1000;

ABC.accountNo = 111;

TransactionDeposit t1;

TransactionWithdraw t2;

t1 = new TransactionDeposit(ABC, 500);

t2 = new TransactionWithdraw(ABC,900);

}

}

1. Repeat the same program using synchronized keyword so that data inconsistence can be avoided.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***