============

File Handling

===========

-> File Handling is Very important area in every programming language

-> Below are the common file operations in the project

1) Create a file

2) Write the data to file

3) Read the data from file

4) Delete the file

-> To perform File operations java language provided one predefined class java.io.File

-> java.io package contains set of classes & interfaces to perform input and output operations

File f = new File (String name);

File f1 = new File (File parent, String child ) ;

-> We have several methods in the file class

boolean createNewFile ( ) : It is used to create a new empty file

boolean mkdir ( ) : It is used to create new empty directory

String[ ] list ( ) : It is used to read the content of the given path

boolean delete ( ) : It is used to delete the file / directory based on given name

isFile ( ) : To check weather it is a file or not

isDirectory ( ) : To check weather it is a directory or not

import java.io.File;

import java.io.IOException;

public class Demo {

public static void main(String... args) throws IOException {

File f = new File("demo.txt");

boolean fstatus = f.createNewFile ( );

System.out.println(fstatus);

File f1 = new File("java.txt");

boolean f1status = f1.createNewFile ( );

System.out.println(f1status);

File f2 = new File("mywork");

boolean f2status = f2.mkdir( );

System.out.println(f2status);

File f3 = new File("data");

f3.mkdir ( );

File f4 = new File(f3, "test.txt");

f4.createNewFile ( );

}

}

// Java program to display all the files and directories in given path

import java.io.File;

import java.io.IOException;

public class Demo {

public static void main(String... args) throws IOException {

File f = new File ("C:\\Usersitashok\\classes\\demo.txt");

String [ ] arr = f.list ( );

for ( String name : arr) {

System.out.println (name);

}

}

}

===============

IO Streams

===============

-> To perform operations on the file we need to use I/O Streams.

-> Using I/O streams we can establish link between java program and file (physical file )

write

java program <--------------------> file

read

java program <-----------------------> file

-> IO streams are divided into 2 types

1) Byte Stream : To read/write binary data (images, audios, videos, pdfs etc...)

2) Character Stream : To read/write character data ( text files )

-> Byte Stream providing 2 types of classes

1) Input Stream Related Classes ( Ex: FileInputStream )

2) Ouput Stream realted classes (Ex: FileOuputStream )

-> Character Stream providing 2 types of classes

1) Reader classes ( Ex: FileReader )

2) Writer classes (Ex: FileWriter )

// Java Program to write the data to a file using FileWriter class

import java.io.\*;

public class Demo {

public static void main(String... args) throws IOException {

FileWriter fw = new FileWriter("data.txt");

fw.write("Hi, good evening");

fw.write("\n"); // it represents new line

fw.write("How are you?");

fw.flush ( );

fw.close( ) ;

}

}

/ Java Program to read file data using BufferedReader

import java.io.\*;

public class Demo {

public static void main(String... args) throws IOException {

FileReader fr = new FileReader("data.txt");

BufferedReader br = new BufferedReader(fr);

String line = br.readLine ( ); // reading first line data

while ( line != null ) {

System.out.println( line );

line = br.readLine ( ) ; // reading next line and re-initialzing line variable

}

}

}

==========================

Serialization & De-Serialization

==========================

-> When we store data in the object, that data will be available if our program is running. If our program got terminated then we will loose our objects and data available in the objects.

-> If we don't want to loose the data even after program got terminated then we should go for Serialization.

-> The process of converting java object into file data in the form of bits and bytes is called as Serialization.

-> The process of converting file data back to java object is called De-Serialization.

Note: To perform Serialization & De-Serialization we have to implement java.io.Serializable interface which is marker interface

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.Serializable;

public class Person implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -100l;

int id;

String name;

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

Person p = new Person();

p.id = 100;

p.name = "Raju";

System.out.println("====Serialization Started ========");

FileOutputStream fos = new FileOutputStream("person.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(p);

oos.flush();

oos.close();

System.out.println("====Serialization completed========");

System.out.println("==========De-Serialization Started==========");

FileInputStream fis = new FileInputStream("person.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

Object object = ois.readObject();

Person p1 = (Person) object;

System.out.println("Id : " + p1.id);

System.out.println("Name : " + p1.name);

ois.close();

System.out.println("==========De-Serialization Ended==========");

}

}

========================

What is SerialVersionUID ?

========================

-> For every .class file JVM will assign one random number that is called as serialVersionUID.

-> When we serialize the object, JVM will assign .class file serialVersionUID to serialized file

-> When we de-serialize JVM will compare serialized file UID and .class file UID. If both ids are matching then only de-serialization will happen otherwise it will throw INvalidClassException.

-> To overcome this problem we can write our own serialVersionUID then jvm will not assign that.

================

transient keyword

================

-> If we have any sensitive / secret data then we shouldn't serialize those fileds.

-> transient keyword is used to ignore variables from serialization process

public class Person implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -100l;

int id;

String name;

String email;

transient String pwd;

}

Note: If we serialize the above person class object, id, name and email will be serialized and pwd will not get serialized because it is transient variable.

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.Serializable;

public class Person implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -100l;

int id;

String name;

String email;

transient String pwd;

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

Person p = new Person();

p.id = 100;

p.name = "Raju";

p.email = "raju@gmail.com";

p.pwd = "raj@123";

System.out.println("====Serialization Started ========");

FileOutputStream fos = new FileOutputStream("person.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(p);

oos.flush();

oos.close();

System.out.println("====Serialization completed========");

System.out.println("==========De-Serialization Started==========");

FileInputStream fis = new FileInputStream("person.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

Object object = ois.readObject();

Person p1 = (Person) object;

System.out.println("Id : " + p1.id);

System.out.println("Name : " + p1.name);

System.out.println("Email : " + p1.email);

System.out.println("Pwd : " + p1.pwd);

ois.close();

System.out.println("==========De-Serialization Ended==========");

}

}