Java – Learn From Home

Assignment – Chapter 7

Concept: IO Streams

Objective: At the end of the assignment, participants will be able to:

- Use Java streams for input and output
- Read and write files
- Serialize object in a file

Problems:

Exercise 1:

Write a java application to find the file name given by the user is available in the directory are not.

Guided Solution:

Step 1: Import a package java.io. *, java.util.* and Create a class named FindFile and write a main() method inside the FindFile class.

Step 2: Create a scanner object to get the file name from the User.

E.g.:

```
Scanner scan = new Scanner(System.in);
System.out.println("Please Enter the file name with Extension :");
String fileName = scan.next();
```

Step 3: Create File class instance and pass the file name given by the user.

```
File f = new File(fileName);
```

Step 4 : call the exists() method using the File instance that was created in the previous step. call this exists() method inside the if() condition.

E.g.:

```
if(f.exists()){
}
```

Step 5: Print "Given File Found" if exists() method returns true else return "Sorry Given File Not Found" in the else block

E.g.:

```
if(f.exists()){
System.out.println("Given File Found");
} else{
System.out.println("Sorry Given File Not Found");
}
```

Step 10: Compile the class using "javac FindFile.java" and Execute as "java FindFile"

Exercise 2:

Write a java application to create a new directory named "MyFolder" using File class.

Guided Solution:

Step 1: Import a package java.io.* and Create a class named CreateDirectory and write a main() method inside the FindFile class.

Step 2: Create File class instance and pass the directory name as "MyFolder".

E.g:

```
String fileName = "MyFolder";
File f = new File(fileName);
```

Step 3: call the mkdir() method using the File instance that was created in the previous step. call the mkdir() method inside the try and catch block.

E.g.:

```
f.mkdir();
```

Step 4: Print "Directory Created"

```
System.out.println("Directory Created");
```

Step 5: Compile the class using "javac CreateDirectory.java" and Execute as "java CreateDirectory"

Exercise 3:

Write a java application that copies the content of a file from file1.txt and write the copied content into the file named file2.txt.Every time when the content is written into the file2.txt the content should get APPENDED into the file.

Guided Solution:

Step 1: Import a package java.io.* and Create a class named CopyFile and write a main() method inside the CopyFile class.

Step 2: Inside the main() method create FileReader and FileWriter as local variables and assign to null.

E.g.:

```
FileReader  fr=null;
FileWriter fw = null;
```

Step 3: Define a try and catch block(IOException) and print the exception msg inside the catch block.

E.g.:

```
try{

}catch(IOException ioe){
  System.out.println(ioe.getMessage());
}
```

Step 4: Create an instance for FileReader and FileWriter varible created before and pass the "file1.txt" and "file2.txt" string value as a constructor argument to the corresponding instances. ensure FieWriter instance takes Boolean as its second arguments for appending the content.

```
fr = new FileReader("file1.txt");
fw = new FileWriter("file2.txt",true);
```

Step 5: Create an character array of size say 25.

```
char[] arr = new char[22];
```

Step 6: Declare and initialize an integer variable to a value 0.

```
int charRead = 0;
```

Step 7: Write a while loop and apply the condition the read the file until the file reaches the EOF(End Of File).

```
while((charRead = fr.read(arr))!=-1){
}
```

Step 8: Inside the while loop call the FileWriter instance variable to invoke the write()

method to write the content into the file2.txt.

Step 9: Close the FileWriter and FileReader class instances.

Step 10: Compile the class using "javac CopyFile.java" and Execute as "java CopyFile"

Exercise 4:

Create a java application to serialize an Employee class object into the “employeeStore.ser” file inside the current directory.

Guided Solution:

Step 1: Import a package java.io.* and Create a class named Employee and implement the Serializable marker interface.

Step 2: Create empID,empName and empSalary variables and their getters and setters functionality inside the Employee class and also create the argumented constructor for this class.

Step 3: Save the Employee class in file named "Employee.java".

Step 4: Create a class named StoreEmployee.

Step 5: Inside this class create the Object of Employee class.

E.g.:
```
Employee emp1 = new Employee(101,"Rajesh",25600);
```

Step 6: Create a try and catch block and inside the try block FileOutputStream and ObjectOutputStream objects corespondingly.

E.g.:
```
FileOutputStream fos = new FileOutputStream("employeeStore.ser",true);
```

ObjectOutputStream oos = new ObjectOutputStream(fos);

Step 7: Use the ObjectOutputStream instance to call the writeObject method and pass the employee Object emp1 as its argument.

oos.writeObject(p);

Step 8: Close the FileOutputStream and ObjectOutputStream objects respectively.

```
fos.close();
oos.close();
```

Step 9: Go to the directory form where the application got executed to check if the file got created and the Employee object got serialized.

Step 10: Compile the class using "javac StoreEmployee.java" and Execute as "java StoreEmployee"

Exercise 5:

Write a Java Application to Convert Given String content into a Byte array and store the Byte array content into a given File.
"This example shows how to write byte content to a file".

Guided Solution:

Step 1: Import a package **java.io.*** and Create a class named **WriteByteFile** and write a **main()** method inside the **WriteByteFile** class.

Step 2: Inside the **main()** method create **FileOutputStream** as local variables and assign it to null.

E.g.:
OutputStream opStream = null;

Step 3: Define a try and catch block(IOException) and print the exception Msg inside the catch block.

```
E.g.:
try{

}catch(IOException ioe){
   System.out.println(ioe.getMessage());
}
```

Step 4: Create String content as given below.

```
String strContent = "This example shows how to write byte content to a file";
```

Step 5: Convert the String content to an byte Array using getBytes() method and store the byte into a byte array.

```
byte[] byteContent = strContent.getBytes();
```

Step 6: Create File class object by passing the name of the file **MyTestFile.txt** to its constructor.

```
File myFile = new File("MyTestFile.txt");
```

Step 7: Apply a condition to check if the given file exists for writing the byte content else create a new file with the given file name.

```
// check if file exist, otherwise create the file before writing
        if (!myFile.exists()) {
           myFile.createNewFile();
        }
```

Step 8: Create an instance for FileOutputStream variable created before and pass the File object as its constructor argument to the corresponding instances.

```
opStream = new FileOutputStream(myFile);
```

Step 9: With the FileOutputStream instance variable invoke the write() method to write the byte array content into the given file and flush the object.

```
opStream.write(byteContent);
opStream.flush();
```

Step 10: Close the FileOutputStream object by calling the close() method inside the finally block.

```
finally{
        try{
           if(opStream != null) opStream.close();
        } catch(Exception ex){

        }
    }
```

Step 11: Compile the class using **"javac WriteByteFile.java"** and Execute as **"java WriteByteFile "**