

Working with Binary OR Byte Oriented Stream classes :

FileOutputStream :

* It is a predefined class in java.io package through which we can create a file (Will override If file is already existing) and can write binary data to the file.

* It comes under binary oriented Stream.

Note : We cannot write text OR character data directly to the FileOutputStream class.

Example :

```
FileOutputStream fos = new FileOutputStream("D:\\new\\Hello.txt");
String str = "Hello Java";
fos.write(str); //Compilation error cannot write character or Text data
```

How to convert character into byte format :

* String class has provided a predefined non static method called getBytes() through which we can convert String data (characters) into byte array.

```
public byte [] getBytes()
```

```
//Program
package com.ravi.file_demo;

import java.util.Arrays;

public class CharacterToByte
{
    public static void main(String[] args)
    {
        String str = "abcdef";

        byte[] data = str.getBytes();
        System.out.println(Arrays.toString(data)); // [97, 98.....]
    }
}
```

WAP to create the file and write binary data to the file :

```
package com.ravi.binary_stream;

import java.io.FileOutputStream;
import java.io.IOException;

public class FileOutputStreamDemo
{
    public static void main(String[] args) throws IOException
    {
        var fos = new FileOutputStream("D:\\new\\India.txt");
        try(fos)
        {
            String str = "Hello India";
            byte[] data = str.getBytes();
            fos.write(data);
            System.out.println("data stored successfully!!!");
        }
    }
}
```

FileInputStream :

* It is a predefined class available in java.io package.

* It comes under binary Stream.

* It is used to read the content of the existing file in binary format.

Method :

```
public int read() :
```

* It is used to read a single character from the source.

* It will return the UNICODE/ASCII value of the character. [A -- 65]

* It will return -1, If the data is not available in the file which represents EOF (End of the file)

```
package com.ravi.binary_stream;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FileInputStreamDemo {

    public static void main(String[] args) throws IOException
    {
        var fin = new FileInputStream("D:\\new\\Hyderabad.txt");
        try(fin)
        {
            int i = 0;

            while(true)
            {
                i = fin.read();
                if(i == -1)
                {
                    break;
                }
                System.out.print((char)i);
            }
        }
    }
}
```

Working with Character Oriented Stream :

In order to perform read and write operation, We can use character Stream also.

Reader and Writer both are the abstract classes which represent super classes for performing all types of input and output operation using character stream.

FileWriter class :

It is a predefined class available in java.io package, By using this class we can create a file and write character data to the file.

By using this class we can directly write String (collection of characters) Or character array to the file.

Actually It is a character oriented Stream where as if we work with FileOutputStream class, It is byte oriented Stream.

```
package com.ravi.character_stream;

import java.io.FileWriter;
import java.io.IOException;

public class WritingCharacter {

    public static void main(String[] args) throws IOException
    {
        FileWriter fw = new FileWriter("D:\\new\\Naresh.txt");

        try(fw)
        {
            String str = "An insitute in Ameerpet and KBHP";
            fw.write(str);

            System.out.println("Data Stored");
        }
    }
}
```

FileReader :

It is a predefined class available in java.io package which will directly read the data in character format, here also we have read() method which will read the data character by character and returns UNICODE value of that character.

```
package com.ravi.character_stream;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ReadingCharacter {

    public static void main(String[] args) throws IOException
    {
        FileReader fr = new FileReader("D:\\new\\Naresh.txt");

        int i = 0;
        while(true)
        {
            i = fr.read();
            if(i == -1)
            {
                break;
            }
            System.out.print((char)i);
        }
    }
}
```

***Serialization and Deserialization :

Serialization :

* It is a technique through which we can write Object data to the file.

* In order to write Object data we should use a predefined class called ObjectOutputStream available in java.io package.

* It provides a non static method writeObject(Object obj).

De-Serialization :

* It is a technique through which we can read the Object data from the file.

* In order to read Object data we should use a predefined class called ObjectInputStream available in java.io package.

* It provides a non static method readObject(), the return type of this method is Object.

Employee.java

Employee.txt

Diagram illustrating the serialization and deserialization process:

The diagram shows a box labeled "Employee.java" containing code for writing object data to a file. An arrow labeled "Storing the Object data in the file (serialization)" points from the box to a file icon labeled "Employee.txt". Another arrow labeled "Reading the object data from file (de-serialization)" points from the file icon back to the box, labeled "ObjectInputStream readObject()".

If we want to perform serialization operation on any object then that object must be Serializable object that means the corresponding class must implements from java.io.Serializable marker interface.

While reading the multiple objects from a file, If we reach to EOF (End of file) then It will generate an exception i.e. java.io.EOFException i.e a checked exception.

If the corresponding class does not implement java.io.Serializable interface then JVM will not allow to perform serialization operation and It will generate an exception i.e. java.io.NotSerializableException

```
//Program on Serialization and De-serialization :
-----
```

```
package com.ravi.serialization_deserialization;

import java.io.Serializable;
import java.time.LocalDate;
import java.util.Scanner;

public class Employee implements Serializable
{
    private static final long serialVersionUID = 1L;

    private transient int id;
    private String name;
    private double salary;
    private LocalDate hireDate;

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

    public static Employee getEmployeeObject()
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Employee Id : ");
        int id = Integer.parseInt(sc.nextLine());

        System.out.print("Enter the Employee Name : ");
        String name = sc.nextLine();

        System.out.print("Enter the Employee Salary : ");
        double salary = Double.parseDouble(sc.nextLine());

        LocalDate date = LocalDate.now();
        return new Employee(id, name, salary, date);
    }

    @Override
    public String toString()
    {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", hireDate=" + hireDate + "]";
    }
}

package com.ravi.serialization_deserialization;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Scanner;

public class SerializableDemo
{
    public static void main(String[] args) throws IOException
    {
        var fout = new FileOutputStream("D:\\new\\Employee.txt");
        var oos = new ObjectOutputStream(fout);
        var sc = new Scanner(System.in);

        try(fout; oos; sc)
        {
            System.out.print("Enter the number of objects : ");
            int noOfObj = Integer.parseInt(sc.nextLine());

            for(int i=1; i<noOfObj; i++)
            {
                Employee employee = Employee.getEmployeeObject();
                oos.writeObject(employee);
            }
            System.out.println("Employee Data stored....");
        }
        catch(Exception e)
        {
            System.out.println("Exception encountered :" + e);
        }
    }
}
```

Employee class variables are declared with transient keyword so they will not be serialized and we will get default value for all the fields.

employeeId -> 0
employeeName -> null
employeeSalary -> null

```
-----
```

```
Employee.java
```

```
Employee.txt
```

Note : In the Employee class variables are declared with transient keyword so they will not be serialized and we will get default value for all the fields.

employeeId -> 0
employeeName -> null
employeeSalary -> null