



CSEg2202

Object Oriented Programming

CHAPTER - 6

File and I/O

Objectives

[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]



[Redacted]

Input/output Basics

-

[Redacted text block]

-

[Redacted text block]

Contd.

- [REDACTED]
 - [REDACTED]
 - [REDACTED]
- [REDACTED]
 - [REDACTED]
 - [REDACTED]
 - [REDACTED]

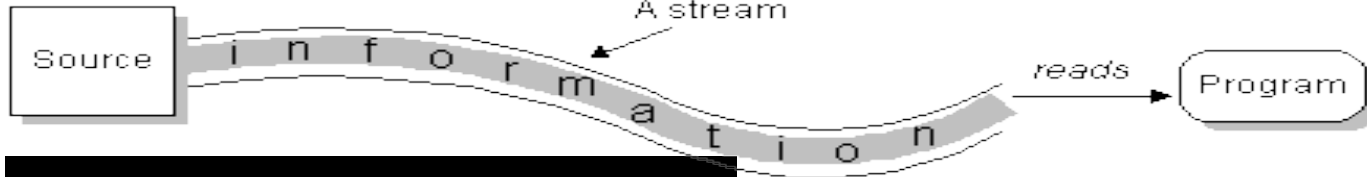
Java I/O Streams

-

-

-

-



-



Contd.

- [REDACTED]
 - [REDACTED]
 - [REDACTED]
 - [REDACTED]
 - [REDACTED]
 - [REDACTED]
 - [REDACTED]
- [REDACTED]
 - [REDACTED]
 - [REDACTED]

Byte vs. Character Streams

[Redacted text block]

[Redacted text block]

[Redacted text block]

[Redacted text block]

Java Classes

Package `java.io` offers classes to connect to streams

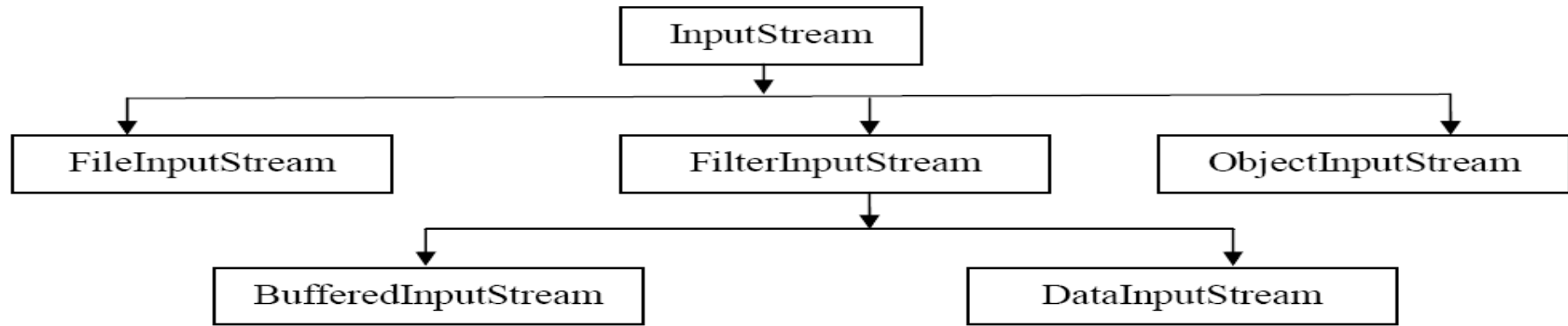
To connect to a stream, instantiate a subclass of one of these abstract super classes:

	input	output
byte	<code>InputStream</code>	<code>OutputStream</code>
character	<code>Reader</code>	<code>Writer</code>

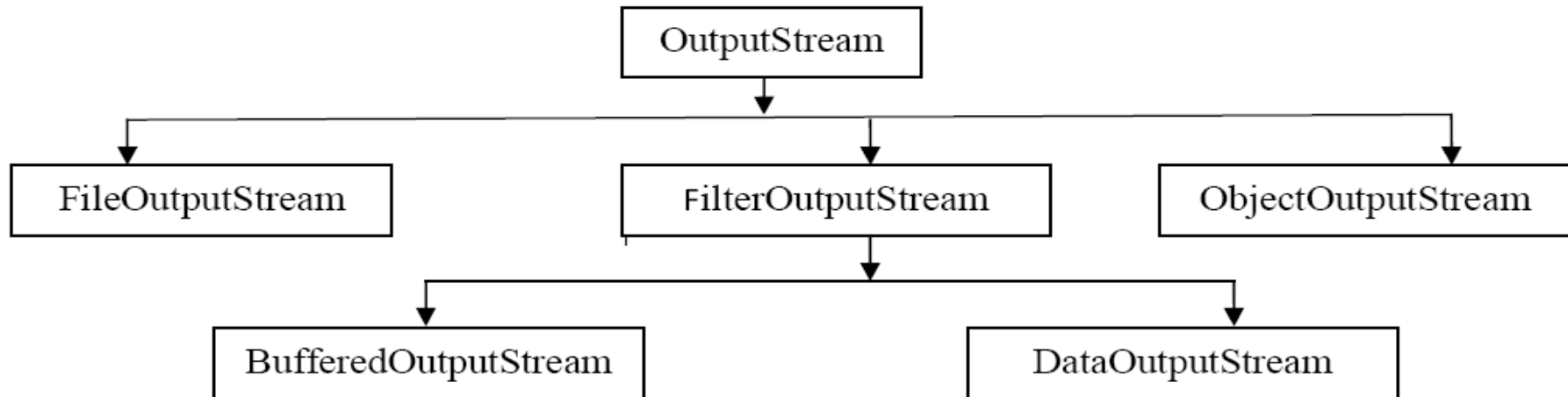
Byte Streams

- Handle data in the form of bits and bytes.
- Byte streams are used to handle any characters (text), images, audio and video files.
- For example, to store an image file (.gif or.jpg), we should go for a byte stream.
- To handle data in the form of 'bytes' the abstract classes: `InputStream` and `OutputStream` are used.
- The important classes of byte streams are:
 - ✓ **`FileInputStream/FileOutputStream`**: They handle data to be read or written to disk files.
 - ✓ **`FilterInputStream/FilterOutputStream`**: They read data from one stream and write it to another stream.
 - ✓ **`ObjectInputStream/ObjectOutputStream`**: They handle storage of objects and primitive data.

Byte Stream Classes



Byte Stream Classes for Reading Data



Byte Stream Classes for Writing Data

Java BufferedOutputStream Class

- Java BufferedOutputStream **class** is used for buffering an output stream.
- It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

```
OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\  
IO Package\\testout.txt"));
```

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

Example of BufferedOutputStream class

```
1. package com.astu;
2. import java.io.*;
3. public class BufferedOutputStreamExample{
4. public static void main(String args[])throws Exception{
5.     FileOutputStream fout=new FileOutputStream("D:\\abc.txt")
6.     BufferedOutputStream bout=new BufferedOutputStream(fout);
7.     String s="Example of BufferedOutputStream class";
8.     byte b[]=s.getBytes();
9.     bout.write(b);
10.    bout.flush();
11.    bout.close();
12.    fout.close();
13.    System.out.println("success");
14. }
15.}
```

Example of Java BufferedInputStream class

```
1.package com.astu;
2.import java.io.*;
3.public class BufferedInputStreamExample{
4.  public static void main(String args[]){
5.    try{
6.      FileInputStream fin=new FileInputStream("D:\\
abc.txt");
7.      BufferedInputStream bin=new BufferedInputStream(fin);

8.      int i;
9.      while((i=bin.read())!=-1){
10.        System.out.print((char)i);
11.      }
12.      bin.close();
13.      fin.close();
14.    }catch(Exception e){System.out.println(e);}
15.  }
```

SequenceInputStream Class

- `SequenceInputStream` is used to read data from multiple `InputStream` objects. It reads data sequentially (one by one).

Java `SequenceInputStream` Class declaration

```
public class SequenceInputStream extends InputStream
```

Example of SequenceInputStream class

```
1.package com.astu;
2.import java.io.*;
3.class InputStreamExample {
4.    public static void main(String args[])throws Exception{
5.        FileInputStream input1=new FileInputStream("D:\\
abc1.txt");
6.        FileInputStream input2=new FileInputStream("D:\\
abc2.txt");
7.        SequenceInputStream inst=new SequenceInputStream(input1, inp
ut2);
8.        int j;
9.        while((j=inst.read())!=-1){
10.            System.out.print((char)j);
11.        }
12.        inst.close();
13.        input1.close();
14.        input2.close();
15.    }
```

ByteArrayStream Class

-

[Redacted text block]

-

[Redacted text block]

-

[Redacted text block]

-

[Redacted text block]

-

[Redacted text block]

Example of Java ByteArrayOutputStream class

```
1.package com.astu;
2.import java.io.*;
3.public class DataStreamExample {
4.public static void main(String args[])throws Exception{
5.    FileOutputStream fout1=new FileOutputStream("D:\\
f1.txt");
6.    FileOutputStream fout2=new FileOutputStream("D:\\
f2.txt");
7.    ByteArrayOutputStream bout=new ByteArrayOutputStream();

8.    bout.write(65);
9.    bout.writeTo(fout1);
10.   bout.writeTo(fout2);
11.   bout.flush();
12.   bout.close();//has no effect
13.   System.out.println("Success...");
14. }
```

Example of Java ByteArrayInputStream class

```
1.package com.astu;
2.import java.io.*;
3.public class ReadExample {
4.    public static void main(String[] args) throws IOException {

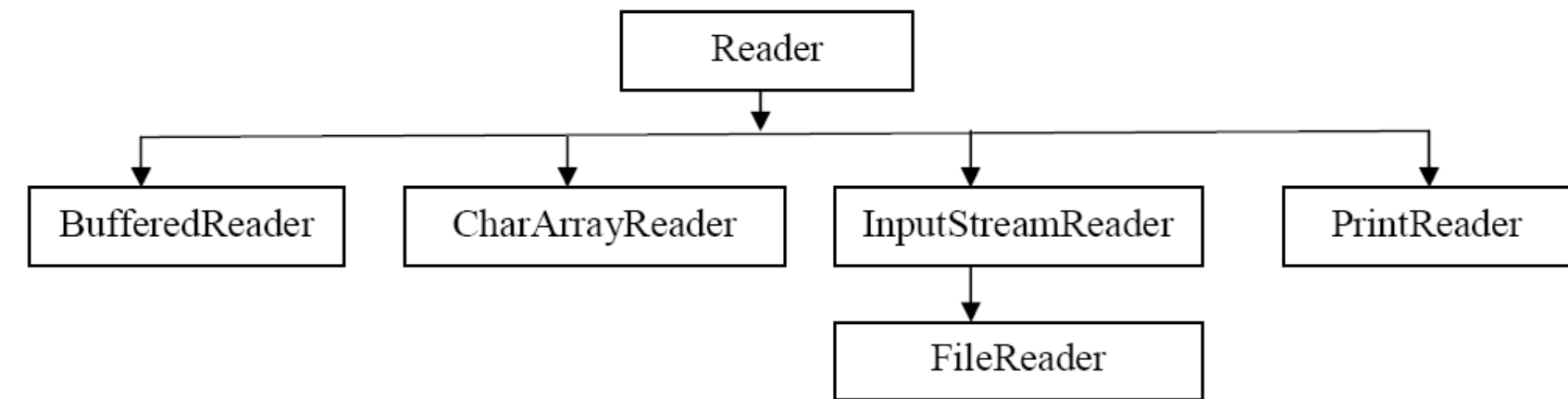
5.        byte[] buf = { 35, 36, 37, 38 };
6.        // Create the new byte array input stream
7.        ByteArrayInputStream byt = new ByteArrayInputStream(buf);

8.        int k = 0;
9.        while ((k = byt.read()) != -1) {
10.            //Conversion of a byte into character
11.            char ch = (char) k;
12.            System.out.println("ASCII value of Character is:" + k + "
; Spec
13.                            ial character is: " + ch);
14.        }
```

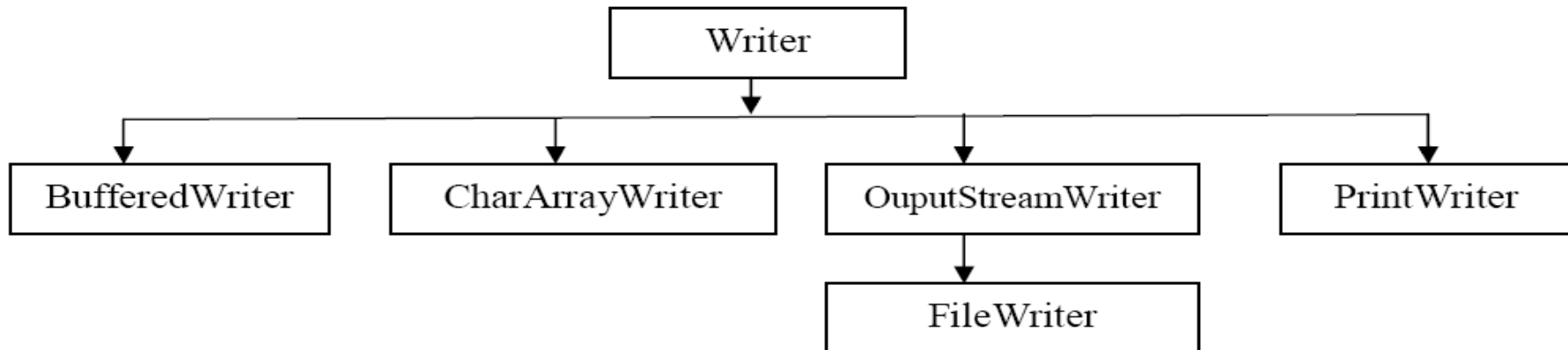
Character or Text Streams

- Handle data in the form of characters.
- Character or text streams can always store and retrieve data in the form of characters (or text) only.
 - ✓ It means text streams are more suitable for handling text files like the ones we create in Notepad.
 - ✓ They are not suitable to handle the images, audio or video files.
- To handle data in the form of 'text', the abstract classes: **Reader** and **Writer** are used.
- The important classes of character streams are:
 - ✓ **BufferedReader/BufferedWriter**: - Handles characters (text) by buffering them. They provide efficiency.
 - ✓ **CharArrayReader/CharArrayWriter**: - Handles array of characters.
 - ✓ **PrintReader/PrintWriter**: - Handle printing of characters on the screen.

Text Stream Classes



Text Stream Classes for Reading Data



Text Stream Classes for Writing Data

Example of Java BufferedWriter

- Let's see the simple example of writing the data to a text file **abc.txt** using Java BufferedWriter.

```
1.package com.astu;
2.import java.io.*;
3.public class BufferedWriterExample {
4.public static void main(String[] args) throws Exception {

5.    FileWriter writer = new FileWriter("D:\\abc.txt");
6.    BufferedWriter buffer = new BufferedWriter(writer);
7.    buffer.write("Example of Java BufferedWriter");
8.    buffer.close();
9.    System.out.println("Success");
10.   }
11.}
```

Example of Java BufferedReader

```
1.package com.astu;
2.import java.io.*;
3.public class BufferedReaderExample {
4.    public static void main(String args[])throws Exception{

5.        FileReader fr=new FileReader("D:\\abc.txt");
6.        BufferedReader br=new BufferedReader(fr);
7.
8.        int i;
9.        while((i=br.read())!=-1){
10.            System.out.print((char)i);
11.        }
12.        br.close();
13.        fr.close();
14.    }
15.}
```

CharArrayReader Class

- The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character array as a reader (stream). It inherits Reader class.

```
1.package com.astu;
2.import java.io.CharArrayReader;
3.public class CharArrayExample{
4.    public static void main(String[] ag) throws Exception {
5.
6.        char[] ary = { 'c', 'h', 'a', 'r', 'a', 'r', 'r', 'a', 'y', 'r', 'e',
7.                        'a', 'd', 'e', 'r' };
8.        CharArrayReader reader = new CharArrayReader(ary);
9.        int k = 0;
10.       while ((k = reader.read()) != -1) {
11.           char ch = (char) k;
12.           System.out.print(ch + " : ");
13.           System.out.println(k);
14.       }
```

Example of CharArrayWriter Class

```
1.package com.astu;
2.import java.io.CharArrayWriter;
3.import java.io.FileWriter;
4.public class CharArrayWriterExample {
5.public static void main(String args[])throws Exception{
6.    CharArrayWriter out=new CharArrayWriter();
7.    out.write("Example of CharArrayWriter");
8.    FileWriter f1=new FileWriter("D:\\a.txt");
9.    FileWriter f2=new FileWriter("D:\\b.txt");
10.   FileWriter f3=new FileWriter("D:\\c.txt");
11.   FileWriter f4=new FileWriter("D:\\d.txt");
12.   out.writeTo(f1);
13.   out.writeTo(f2);
14.   out.writeTo(f3);
15.   out.writeTo(f4);
16.   f1.close();
17.   f2.close();
18.   f3.close();
19.   f4.close();
20.   System.out.println("Success...");
21. }
```

PrintWriter Example

```
1.package com.astu;
2.import java.io.File;
3.import java.io.PrintWriter;
4.public class PrintWriterExample {
5.    public static void main(String[] args) throws Exception {
6.        //Data to write on Console using PrintWriter
7.        PrintWriter writer = new PrintWriter(System.out);
8.        writer.write("Example of print writer.");
9.        writer.flush();
10.       writer.close();
11.       //Data to write in File using PrintWriter
12.       PrintWriter writer1 = null;
13.       writer1 = new PrintWriter(new File("D:\\testout.txt"));
14.       writer1.write("example of print writer");
15.       writer1.flush();
16.       writer1.close();
17.   }
18.}
```

File

- A file represents organized collection of data.
- Data is stored permanently in the file.
- Once data is stored in the form of a file we can use it in different programs.

Java FileInputStream /output Stream Class

- Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class .
- You can also read/write character-stream data. But, for reading / writing streams of characters, it is recommended to use FileReader class.

Example of Java FileOutputStream

```
1.import java.io.FileOutputStream;
2.public class FileOutputStreamExample {
3.    public static void main(String args[]){
4.        try{
5.
6.            FileOutputStream fout=new FileOutputStream("D:\\abc.txt");
7.            fout.write(65);
8.            fout.close();
9.            System.out.println("success...");
10.        }catch(Exception e)
11.        {System.out.println(e);}
12.    }
```

Example of Java FileInputStream

```
1.import java.io.FileInputStream;
2.public class DataStreamExample {
3.    public static void main(String args[]){
4.        try{
5.
6.            FileInputStream fin=new FileInputStream("D:\\
7.                abc.txt");
8.            int i=fin.read();
9.            System.out.print((char)i);
10.
11.            fin.close();
12.        }catch(Exception e)
13.        {System.out.println(e);}
14.    }
```

Note: Before running the code, a text file named as “**abc.txt**” is required to be created.

Example of Java FileWriter

- In this example, we are writing the data in the file testout.txt using Java FileWriter class.

```
1.package com.astu;
2.import java.io.FileWriter;
3.public class FileWriterExample {
4.    public static void main(String args[]){
5.        try{
6.            FileWriter fw=new FileWriter("D:\\abc.txt");
7.            fw.write("FileWriter Example.");
8.            fw.close();
9.        }catch(Exception e){System.out.println(e);}
10.       System.out.println("Success...");
11.    }
12.}
```

Example of Java FileReader

- In this example, we are reading the data from the text file **abc.txt** using Java FileReader class.

```
1.package com.astu;
2.
3.import java.io.FileReader;
4.public class FileReaderExample {
5.    public static void main(String args[])throws Exception{
6.        FileReader fr=new FileReader("D:\\
testout.txt");
7.        int i;
8.        while((i=fr.read())!=-1)
9.            System.out.print((char)i);
10.        fr.close();
11.    }
12.}
```

Object Serialization

- Serialization is the process of storing object contents into a file.
- The class whose objects are stored in the file should implement "Serializable" interface of java.io package.
 - ✓ Serializable interface is an empty interface without any members and methods, such an interface is called 'marking interface' or 'tagging interface'.
 - ✓ Marking interface is useful to mark the objects of a class for a special purpose.
 - ✓ For example, 'Serializable' interface marks the class objects as 'serializable' so that they can be written into a file.
 - ✓ If serializable interface is not implemented by the class, then writing that class objects into a file will lead to `NotSerializableException`.
- static and transient variables cannot be serialized.
- De-serialization is the process of reading back the objects from a file.

Example

```
1.import java.io.Serializable;
2.public class Student implements Serializable{
3.    int id;
4.    String name;
5.    public Student(int id, String name) {
6.        this.id = id;
7.        this.name = name;
8.    }
9.}
```

- In the above example, **Student** class implements Serializable interface. Now its objects can be converted into stream. The main class implementation of is showed in the next code.

ObjectOutput/Input Stream class

- The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream.
- Only objects that support the java.io.Serializable interface can be written to streams.
- The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object.
- An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.
- The readObject method is responsible for reading and restoring the state of the object for its particular class using data written to the stream by the corresponding writeObject method.

Example of Java Serialization

```
1.import java.io.*;
2.class Persist{
3. public static void main(String args[]){
4. try{
5. Student s1 =new Student(101,"John");
6. //Creating stream and writing the object
7. FileOutputStream fout=new FileOutputStream("f.txt");
8. ObjectOutputStream out=new ObjectOutputStream(fout);
9. out.writeObject(s1);
10. out.flush();
11. //closing the stream
12. out.close();
13. System.out.println("success");
14. }catch(Exception e){System.out.println(e);}
15. }
16.}
```

Example of Java Deserialization

```
1.import java.io.*;
2.class Depersist{
3. public static void main(String args[]){
4.     try{
5.         //Creating stream to read the object
6.         ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.
txt"));
7.         Student s=(Student)in.readObject();
8.         //printing the data of the serialized object
9.         System.out.println(s.id+" "+s.name);
10.        //closing the stream
11.        in.close();
12.    }catch(Exception e){System.out.println(e);}
13. }
14.}
```

File Class

- File class of java.io package provides some methods to know the properties of a file or a directory.
- To use the File class, create an object of the class, and specify the filename or directory name:

```
import java.io.File; // Import the File class
File myObj = new File("filename.txt"); // Specify the
filename
```

File Class

File class Methods:

Methods	Description
boolean isFile ()	Returns true if the File object contains a filename, otherwise false
boolean isDirectory ()	Returns true if the File object contains a directory name
boolean canRead ()	Returns true if the File object contains a file which is readable
boolean canWrite ()	Returns true if the File object contains a file which is writable
boolean canExecute ()	Returns true if the File object contains a file which is executable
Boolean exists ()	Returns true when the File object contains a file or directory which physically exists in the computer.
String getParent ()	Returns the name of the parent directory
String getPath ()	Gives the name of directory path of a file or directory
String getAbsolutePath ()	Gives the fully qualified path
long length ()	Returns a number that represents the size of the file in bytes
boolean delete ()	Deletes the file or directory whose name is in File object
boolean createNewFile ()	Automatically crates a new, empty file indicated by File object, if and only if a file with this name does not yet exist.

Example

```
import java.io.*;
class FileProp
{   public static void main(String args[])
    {   String fname = "Myfile";
        File f = new File (fname);
        System.out.println ("File name: " + f.getName());
        System.out.println ("Path:" + f.getPath ());
        System.out.println ("Absolute Path:" + f.getAbsolutePath ());
        System.out.println ("Parent:" + f.getParent ());
        System.out.println ("Exists:" + f.exists ());
        if (f.exists()){
            System.out.println ("Is writable: " + f.canWrite ());
            System.out.println ("Is readable: " + f.canRead ());
            System.out.println ("Is executable: " + f.canExecute ());
            System.out.println ("Is directory: " + f.isDirectory ());
            System.out.println ("File size in bytes: " + f.length ());
        }
    }
}
```

THE END



Adama Science and Technology University
School of Electrical Engineering And Computing

2023 G.C