Core Java 8 and Development Tools

Lesson 13: File IO

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to

- Understand concept of Java I/O API
- Implements byte and character streams to perform I/O
- Work with utility classes like File and Path

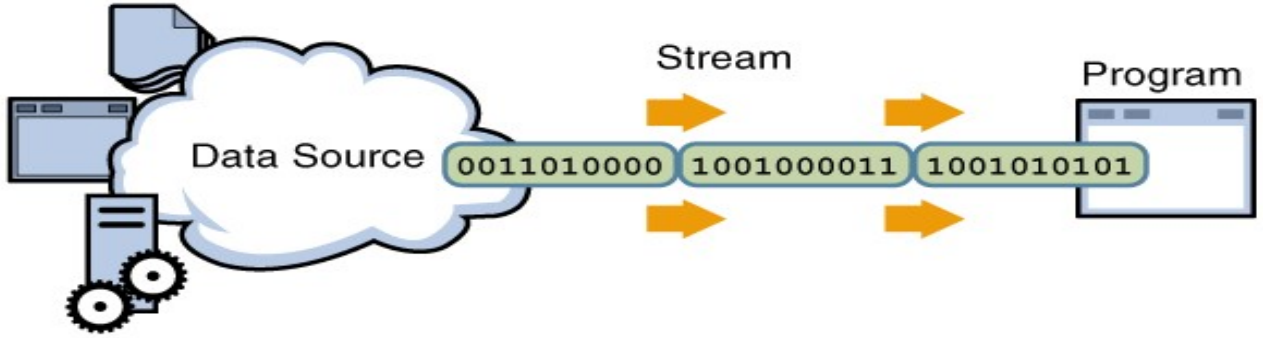
13.1 : Overview of I/O Streams

Overview

Most programs need to access external data.

Data is retrieved from an input source. Program results are sent to output destination.

Figure 7-1: A program uses an input stream to read data from a source, one item at a time



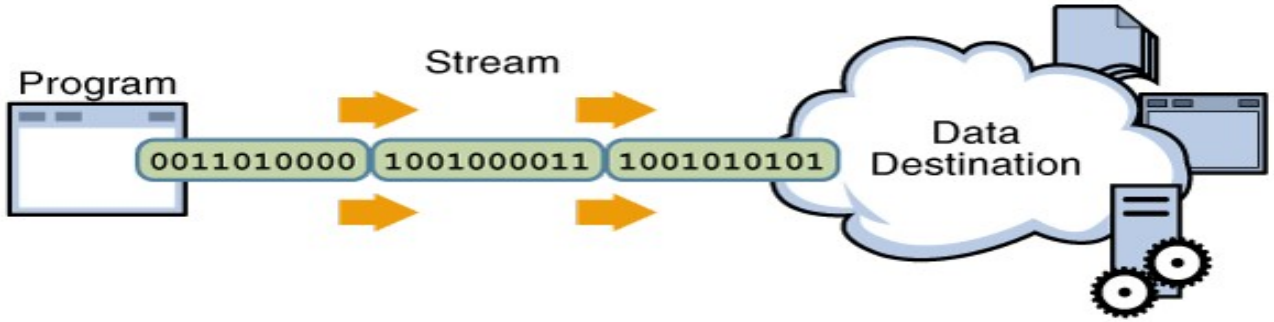


Figure 7-2:A program uses an output stream to write data to a destination, one item at time

13.1: Overview of I/O Streams
What is a Stream?

Stream:

- Abstraction that consumes or produces information.
- Linked to source and destination.
- Implemented within class hierarchies defined in java.io package.
- An input stream acts as a source of data.
- An output stream acts as a destination of data.

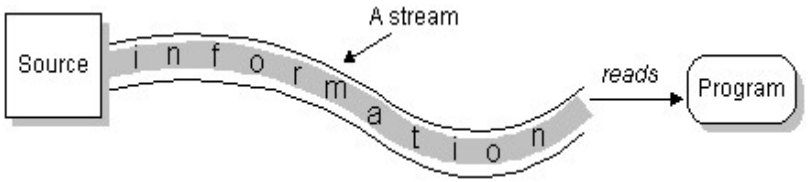


Figure 7-3: (a) Input Stream

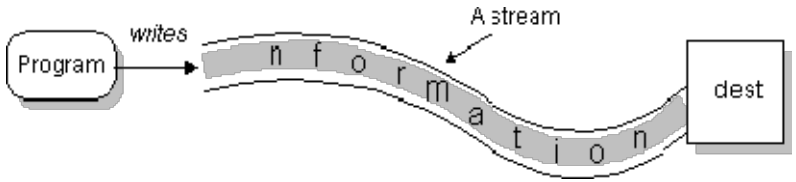


Figure 7-3:(b) Output stream

13.2: Types of Streams
Different Types of I/O Streams

Byte Streams: Handle I/O of raw binary data.

Character Streams: Handle I/O of character data. Automatic translation handling to and from a local character.

Buffered Streams: Optimize input and output with reduced number of calls to the native API.

Data Streams: Handle binary I/O of primitive data type and String values.

Object Streams: Handle binary I/O of objects.

Scanning and Formatting: Allows a program to read and write formatted text.

13.3: Byte Stream I/O Hierarchy
Byte Stream I/O Hierarchy

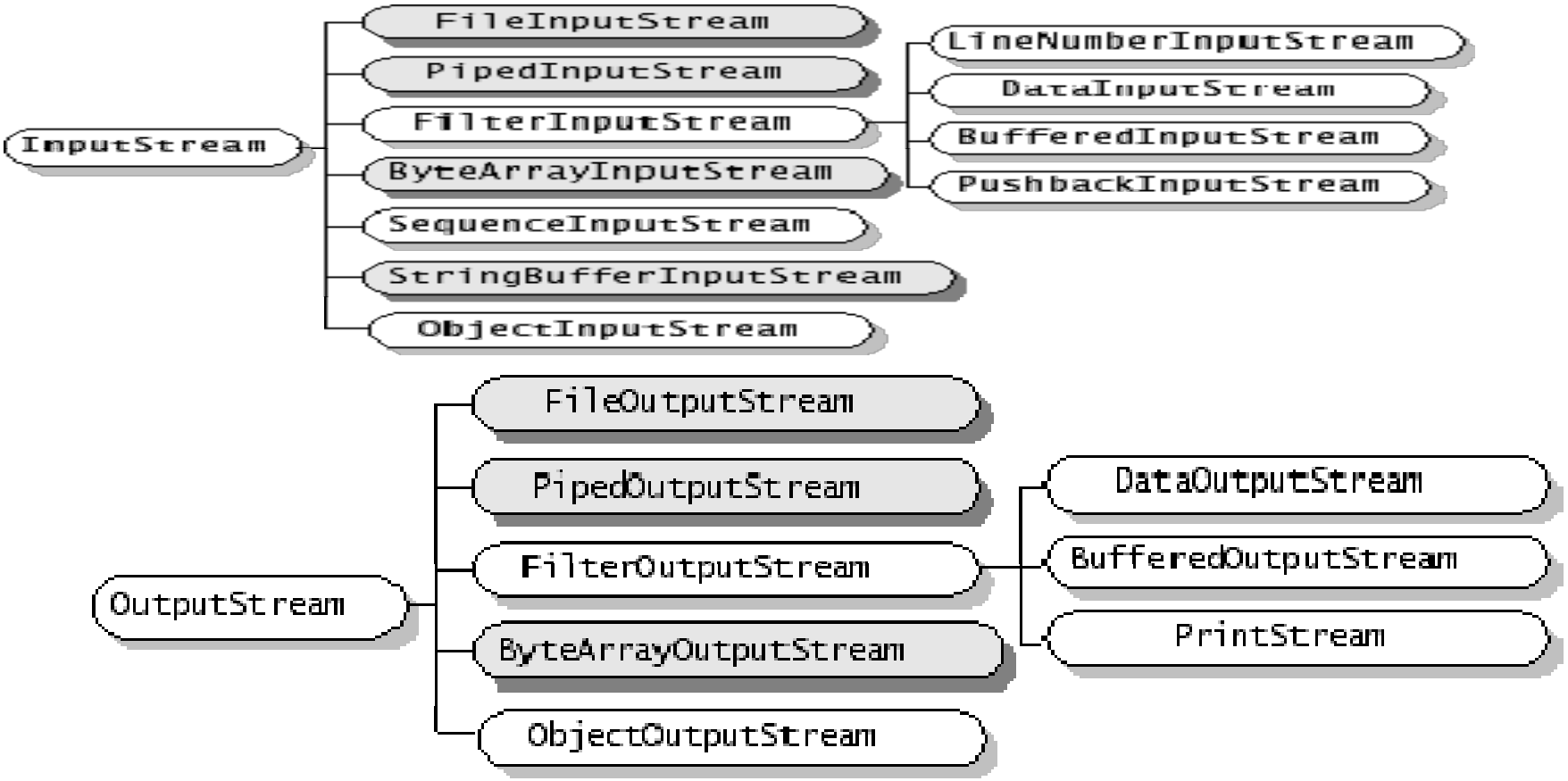


Figure 7-4:Byte-stream I/O hierarchy

13.3: Byte Stream I/O Hierarchy
Methods of InputStream Class

Method	Description
close()	Closes this input stream and releases any system resources associated with the stream.
int read()	Reads the next byte of data from the input stream.
int read(byte[] b)	Reads some number of bytes from the input stream and stores them into the buffer array *b*.
int read(byte[] b, int off, int len)	Reads up to *len* bytes of data from the input stream into an array of bytes.

Table 7-1: Methods of class InputStream

13.3: Byte Stream I/O Hierarchy
Methods of OutputStream Class

Method	Description
close()	Closes this output stream and releases any system resources associated with this stream.
flush()	Flushes this output stream and forces any buffered output bytes to be written out.
write(byte[] b)	Writes *b.length* bytes from the specified byte array to this output stream.
write(byte[] b, int off, int len)	Writes *len* bytes from the specified byte array starting at offset off to this output stream.
write(int b)	Writes the specified byte to this output stream.

Table 7-2: Methods of class OutputStream

13.3: Byte Stream I/O Hierarchy
Input Stream Subclasses

Classname	Description
DataInputStream	A filter that allows the binary representation of java primitive values to be read from an underlying inputstream
BufferedInputStream	A filter that buffers the bytes read from an underlying input stream. The buffer size can be specified optionally.
FilterInputStream	Superclass of all input stream filters. An input filter must be chained to an underlying inputstream.
ByteArrayInputStream	Data is read from a byte array that must be specified
FileInputStream	Data is read as bytes from a file. The file acting as the input stream can be specified by File object, or as a String
PushBackInputStream	A filter that allows bytes to be "unread " from an underlying stream. The number of bytes to be unread can be optionally specified.
ObjectInputStream	Allows binary representation of java objects and java primitives to be read from a specified inputstream.
PipedInputStream	It reads many bytes from PipedOutputStream to which it must be connected.
SequenceInputStream	Allows bytes to be read sequentially from two or more input streams consecutively.

13.3: Byte Stream I/O Hierarchy
The predefined streams

The java.lang.System class encapsulates several aspects of the run-time environment.

Contains three predefined stream variables: in, out & err.

These fields are declared as public and static within System.

- *System.out* :refers to the standard output stream
- *System.err* :refers to standard error stream
- *System.in* : refers to standard input

13.3: Byte Stream I/O Hierarchy
Example : Reading Console input

```
import java.io.*;
class ReadKeys {
 public static void main (String args[]) {
        StringBuffer sb = new StringBuffer();
        char c;
     System.out.println("Enter a String:");
        try {
              while((c =(char)System.in.read()) != '\n')
                   sb.append(c);
      }catch(Exception e){
            System.out.println("Error while reading" + e.getMessage()); }
  String s = new String(sb);
  System.out.println("You entered : " + s);      }}
```

13.3: Byte Stream I/O Hierarchy
Example: FileInputStream & FileOutputStream

```
class CopyFile  {
      FileInputStream fromFile;  FileOutputStream toFile;
      public void init(String arg1, String arg2) {       //pass file names
            try{
               fromFile = new FileInputStream(arg1);
               toFile = new FileOutputStream(arg2);
            } catch (Exception fnfe) {…}
      }
     public void copyContents() {// copy bytes
       try {
             int i = fromFile.read();
             while ( i != -1) {                          //check the end of file
                   toFile.write(i);
                   i = fromFile.read(); }
      } catch (IOException ioe) { System.out.println("Exception: " + ioe);}
  }
```

13.3: Byte Stream I/O Hierarchy
Demo : FileInput/OutputStream

Execute:

- ReadKeys.java
- CopyFile.java program

13.4: Character Stream Hierarchy
Character Stream Hierarchy

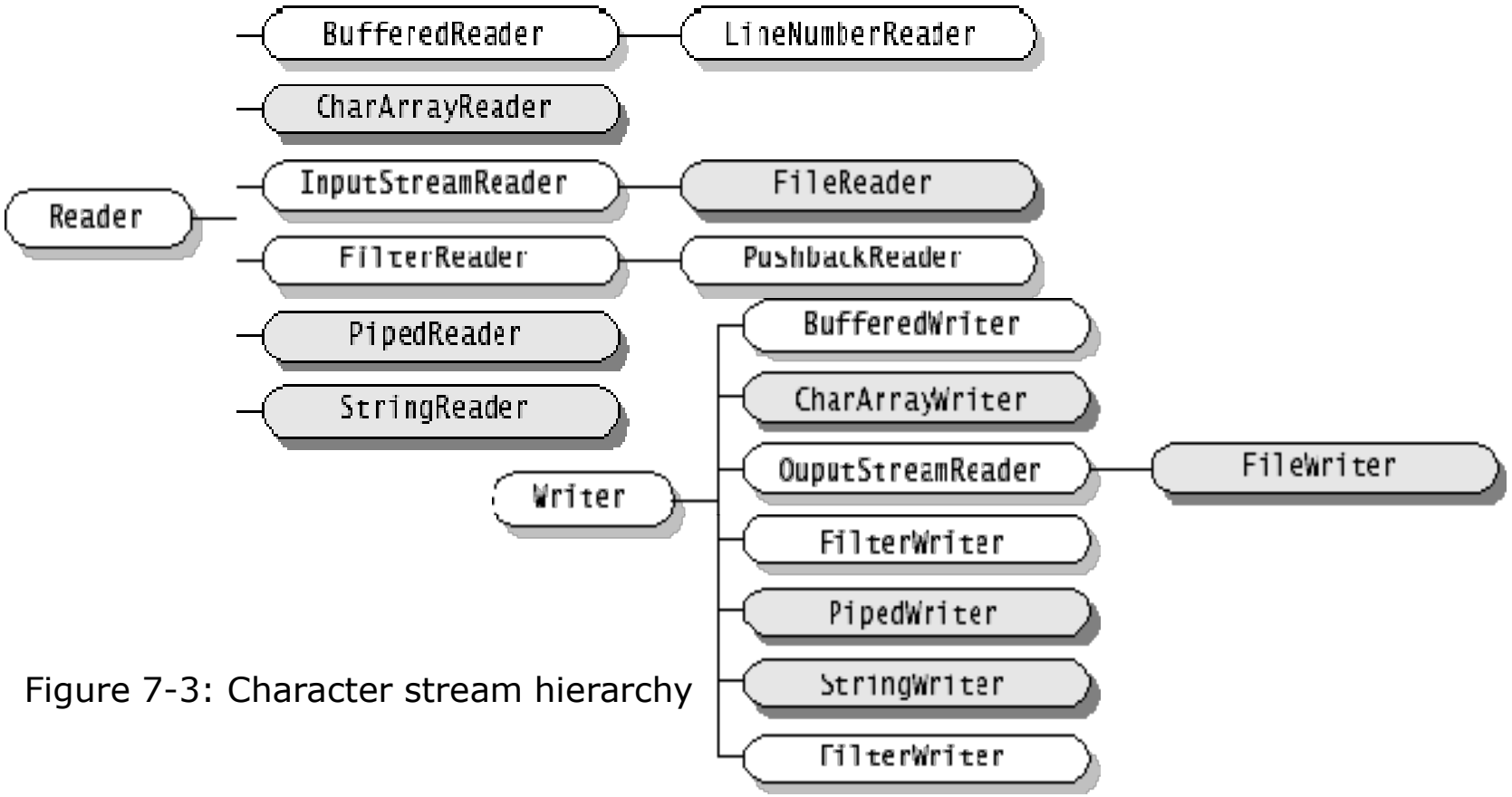


Figure 7-3: Character stream hierarchy

13.4: Character Stream Hierarchy
Reader Class Methods

Method	Description
int read() throws IOException	reads a byte and returns as an int
int read(char b[])throws IOException	reads into an array of chars b
int read(char b[], int off, int len) throws IOException	reads *len* number of characters into char array *b*, starting from offset *off*
long skip(long n) throws IOException	Can skip n characters.

Table 7-4: Reader Methods

13.4: Character Stream Hierarchy
Writer Class Methods

Method	Description
void write(int c)throws IOException	writes a byte.
void write(char b[])throws IOException	writes from an array of chars b
void write(char b[], int off, int len) throws IOException	writes len number of characters from char array b, starting from offset off
void write(String b, int off, int len) throws IOException	writes len number of characters from string b, starting from offset off

Table 7-5: Writer Methods

13.4: Character Stream Hierarchy
Example: FileReader, FileWriter Classes

```
public class CopyCharacters {
   public static void main(String[] args) throws IOException {
 try(FileReader inputStream = new FileReader("sampleinput.txt");
       FileWriter outputStream = newFileWriter("sampleoutput.txt"))
{
            int c;
            while ((c = inputStream.read()) != -1) {
                 outputStream.write(c);
            }
  } catch(IOException ex) {
              System.out.println(ex.getMessage());
   }
} }
```

13.5: Buffered Stream
Buffered Input Output Stream

An unbuffered I/O means each read or write request is handled directly by the underlying OS.

- Makes a program less efficient.
  - Each such request often triggers disk access, network activity, or some other relatively expensive operation.

Java's buffered I/O Streams reduce this overhead.

- Buffered streams read/write data from a memory area known as a buffer; the native input API is called only when the buffer is empty.

13.5: Buffered Stream
Using buffered streams

A program can convert a unbuffered stream into buffered using the *wrapping idiom:*

- Unbuffered stream object is passed to the constructor of a buffered stream class.
- Example

```
inputStream = new BufferedReader(new FileReader(“input.txt"));
outputStream = new BufferedWriter(new FileWriter(“output.txt"));
```

13.5: Buffered Stream
Example of Buffered stream

```
class LineNumberReaderDemo{
        public static void main(String args[]) {
                String s;
                try(FileReader fr = new FileReader("names.txt");
                        BufferedReader br = new BufferedReader(fr);
                        LineNumberReader lr = new LineNumberReader(br);)   {
                        while((s = lr.readLine()) != null)
                                System.out.println(lr.getLineNumber()+" " +s);
                } catch (IOException e) {
                        System.out.println(e.getMessage());
                }
} }
```

Names.txt contains
Anita
Bindu
Cindy
Diana

Output is:
1 Anita
2 Bindu
3 Cindy
4 Diana

13.5: Buffered Stream
Demo: File Reader / File Writer

Execute the

- LineNumberReaderDemo.java
- CharEncode.java

13.6: File class
The File Class

File class doesn’t operate on streams

Represents the pathname of a file or directory in the host file system

Used to obtain or manipulate the information associated with a disk file, such as permissions, time, date, directory path etc

An object of File class provides a handle to a file or directory and can be used to create, rename or delete the entry

13.6: File class
The File Class

Some methods

- canRead()
- exists()
- isFile()
- isDirectory()
- getAbsolutePath()
- getName()
- getPath()
- getParent()
- length() : returns length of file in bytes as long
- lastModified()
- mkdir()
- list() : obtain listings of directory contents

13.6: File class
The File Class

```
class FileDemo  {
    String fname;
    public static void main(String args[]) {
        String fname = args[0];
        File f = new File(fname);
        System.out.println("File name : "+f.getName());
        System.out.println("Parent dir name : "+f.getParent());
        System.out.println("Absolute path name : "+f.getAbsolutePath());
        System.out.println("File modified last :
                                              "+String.valueOf(f.lastModified()));
        System.out.println("File length : "+f.length());
        System.out.println("File Readable? : " + (f.canRead()? "true":"false"));
     } }
```

13.7: Exploring NIO
Path Interface

Java 7 provides new improved features over traditional File class

Files and directories in file system can be uniquely identified by Path

A path can be absolute or relative

Paths class can be used to create a path reference

```
Path javaHome = Paths.get("C:/Program Files/Java/jdk1.8.0_25");
System.out.println(javaHome.getNameCount()); //3 (doesn't count root)
System.out.println(javaHome.getRoot()); // C:\
System.out.println(javaHome.getName(0));// Program Files
System.out.println(javaHome.getName(1)); // Java
System.out.println(javaHome.getFileName()); //jdk1.8.0_25
System.out.println(javaHome.getParent()); //C:\Program Files\Java
```

13.7: Exploring NIO
Files Class

Introduced in java.nio.file for better file and directory manipulation

- File/Directory creation and deletion
- Perform different checks with File/Directory
- Used to create streams objects

Method	Meaning
createFile	Used to create a file
createDirectory	Used to create a directory
delete	Used to delete the file/directory
deleteIfExists	Check before deleting file/directory
newDirectoryStream	Used to fetch directory contents
copy	Copies the file/directory
move	Moves the file/directory
readAllLines/readAllBytes	Used to read file in stream
write	Used to write in file

13.7: Exploring NIO
Demo: Path and Files

Execute the

- PathDemo.java
- ListingDirectory.java
- ListingFile.java

13.8:Object Stream
Object Input Stream, Object Output Stream

Object streams support I/O of objects:

- Support I/O of primitive data types.
- Object has to be *Serializable* type.
- *Object Classes:* ObjectInputStream, ObjectOutputStream
  - Implement ObjectInput and ObjectOutput, which are subinterfaces of DataInput and DataOutput.
- An object stream can contain a mixture of primitive and object values.

13.8: Object stream
Serializing Objects

Object Serialization:

- Process to read and write objects.
- Provides ability to read or write a whole object to and from a raw byte stream.
- Use object serialization in the following ways:
  - Remote Method Invocation (RMI): Communication between objects via sockets.
  - Lightweight persistence: Archival of an object for use in a later invocation of the same program.

13.8: Objects stream
Example : Object Serialization

```
class Student implements Serializable{
    int roll;
    String sname;
    public Student(int r, String s){
          roll = r;
          sname = s;     }
     public String toString(){
                    return "Roll no is : "+roll+"   Name is : "+sname;
    } }
```

```
public class demo{
    public static void main(String args[]){
    try{ Student s1 = new Student (100,"Varsha");
         System.out.println("s1 object : "+s1);
```

13.8: Objects stream
Example: Object Serialization (contd..)

```
FileOutputStream fos = new FileOutputStream("student");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(s1);
oos.flush();
oos.close();
  }   catch(Exception e){  }
  try{
Student s2;
FileInputStream fis = new FileInputStream("student");
ObjectInputStream ois = new ObjectInputStream(fis);
s2 = (Student)ois.readObject();
ois.close();
System.out.println("s2 object : "+s2);   }
catch(Exception e){  }  }
```

13.8: Objects stream
Demo: Object Serialization

Execute the :

- Student.java and ObjectSerializationDemo.java
- EmpObjectSerializationDemo.java

Lab : Files IO

Lab 8: Files IO

13.9: Best Practices in I/O
Best Practices in I/O

Always close streams:

```
try{
 file = new FileOutputStream( "emp.ser" );
 OutputStream buffer = new BufferedOutputStream( file);
 ObjectOutput output = new ObjectOutputStream( buffer );
  try{ output.writeObject(emp); }
 finally{ output.close(); } }
```

- Use buffering when reading and writing text files.
- FileInputStream and DataInputStream are very slow.

13.9: Best Practices in I/O
Best Practices in I/O (contd..)

Do not implement Serializable unless needed.

Serialization and Subclassing

Summary

In this lesson you have learnt:

- Different types of I/O Streams supported by Java
- Important classes in java.io package
- Object Serialization
- Best Practices in Java I/O

Review Question

Question 1: What is a buffer?

- **Option 1 :** Section of memory used as a staging area for input or output data.
- **Option 2 :** Cable that connects a data source to the bus.
- **Option 3 :** Any stream that deals with character IO.
- **Option 4 :** A file that contains binary data.

Question 2: Can data flow through a given stream in both directions?

- True
- False

Question 3: _____ is the name of the abstract base class for streams dealing with *character input*