Java I/O

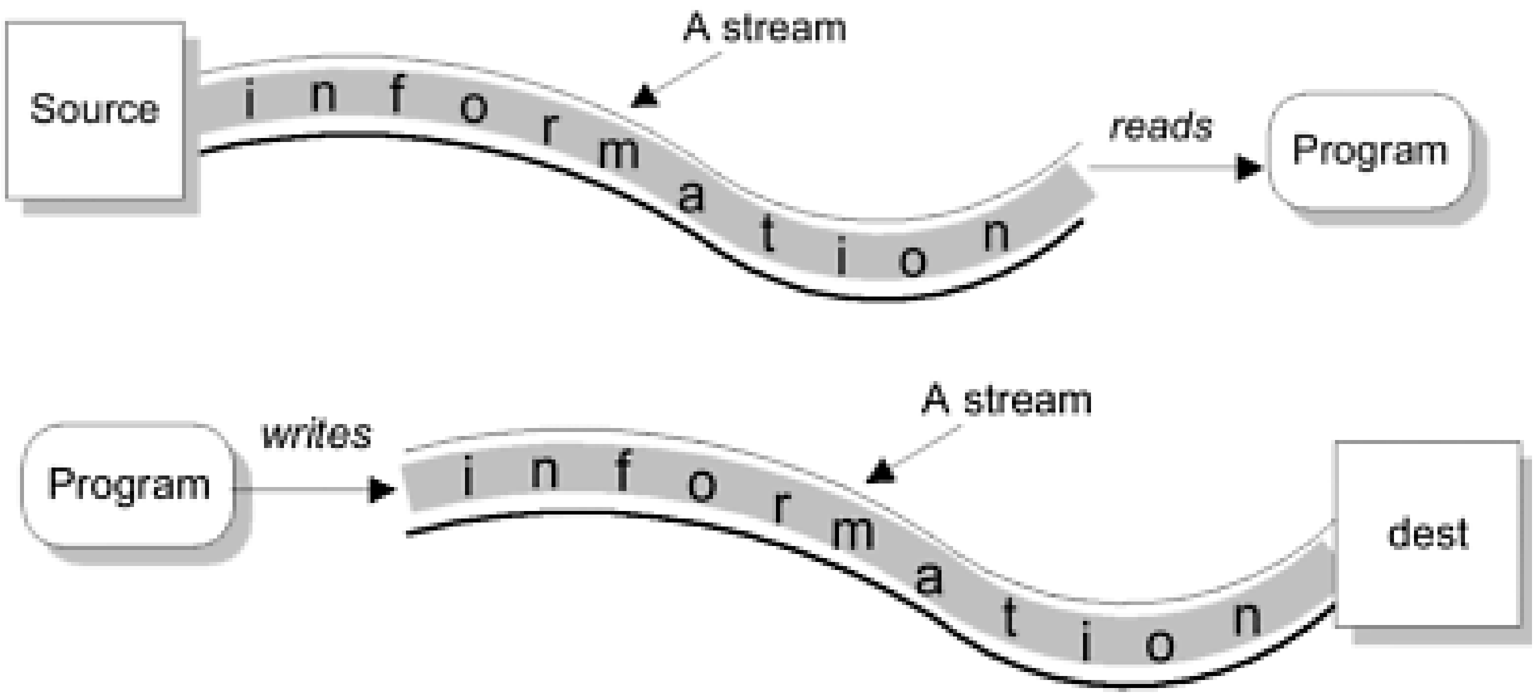
Banu Prakash C

Objectives

- Understanding which streams to use for character-based and byte-based streams.
- Read from a text file and write to a text file.
- Understand wrapper stream classes for providing additional higher level functionality.
- Understand how to read the data from keyboard using stream classes.
- Understand java serialization.

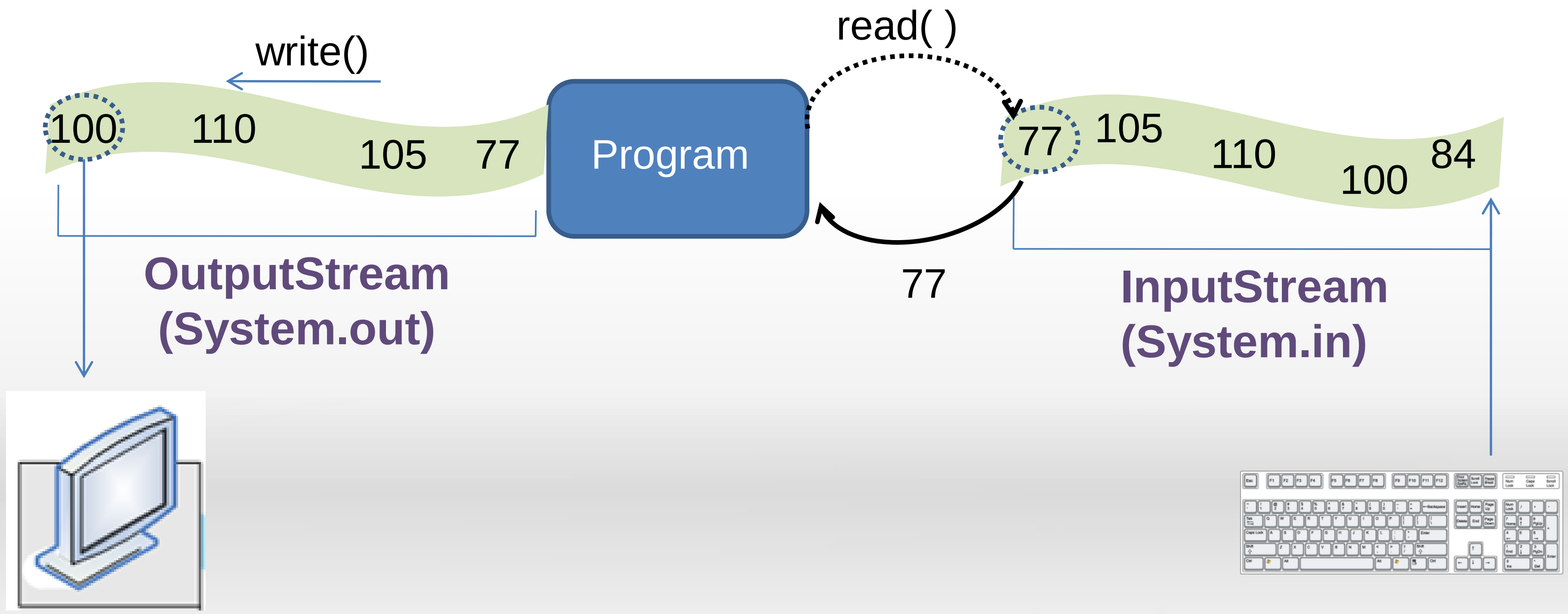
What is a Stream?

- A stream represents a flow of data, or a channel of communication with a writer at one end and a reader at the other.
- All fundamental I/O in java is based on streams.



Input Stream and Output Stream

- In Java there are different streams for reading and writing.
- Use Output Stream to write data from an application.
- Use Input Stream to read data into an application.

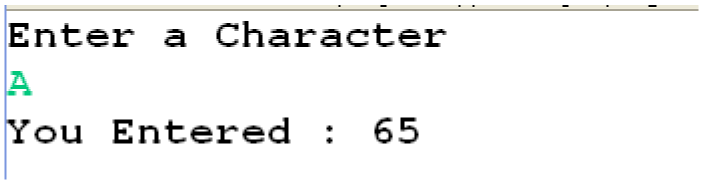


Console I/O

- System.out : out is an object of type PrintStream for outputting data to console.
- System.in : in is an object of type InputStream which allows reading of raw bytes from keyboard.

```
/*
 * OutputStream to output text to Console (Monitor)
 */
System.out.println("Enter a Character");
/*
 * read bytes from keyboard.
 * get UNICODE value of character typed.
 */
int b = System.in.read();

System.out.println("You Entered : " + b);
```

Output Screen:

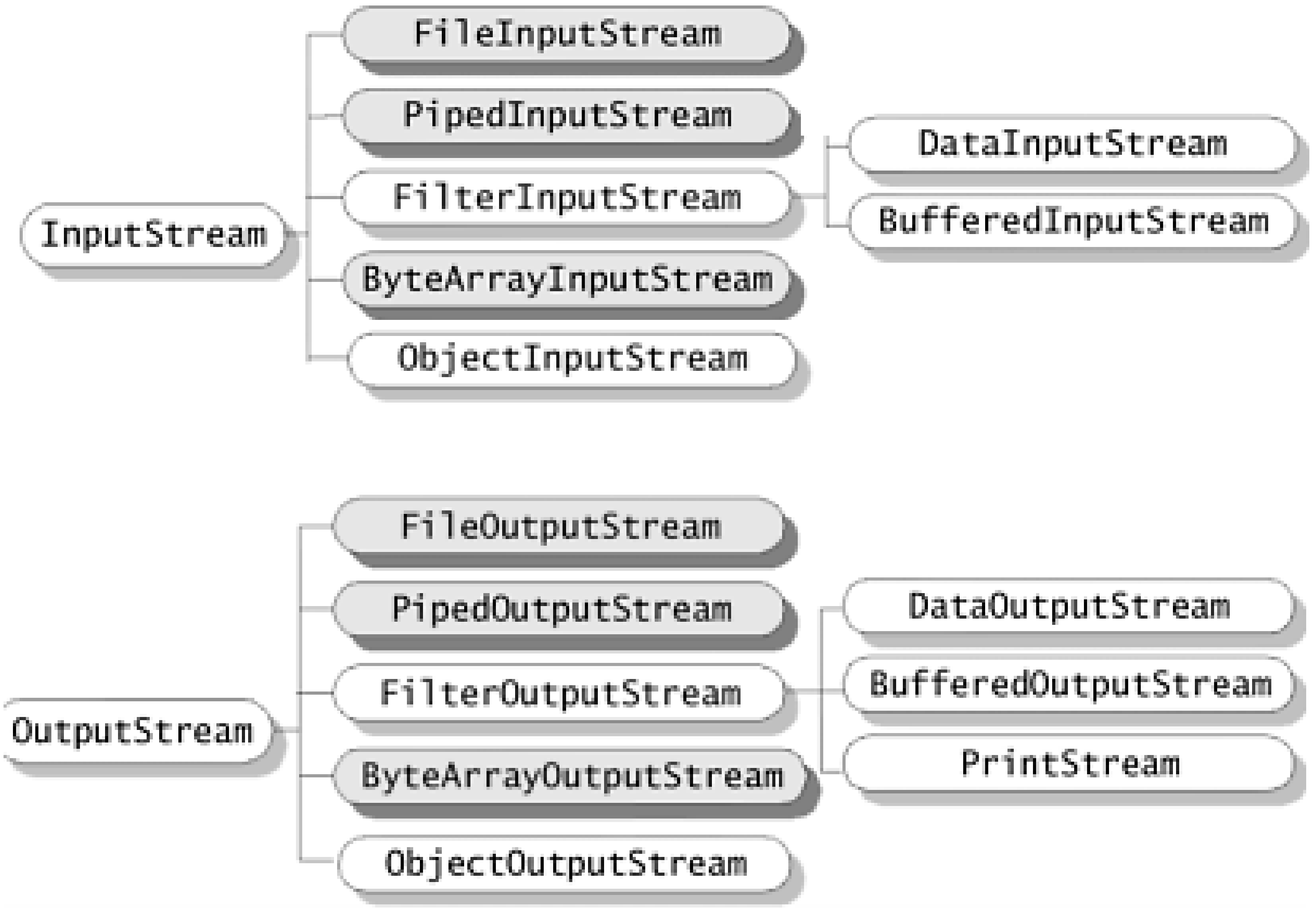


```
Enter a Character
A
You Entered : 65
```

Byte and Character streams

- Byte streams that are subclasses of InputStream read 8-bit bytes.
- Byte streams that are subclasses of OutputStream writes 8-bit bytes

Byte streams hierarchy



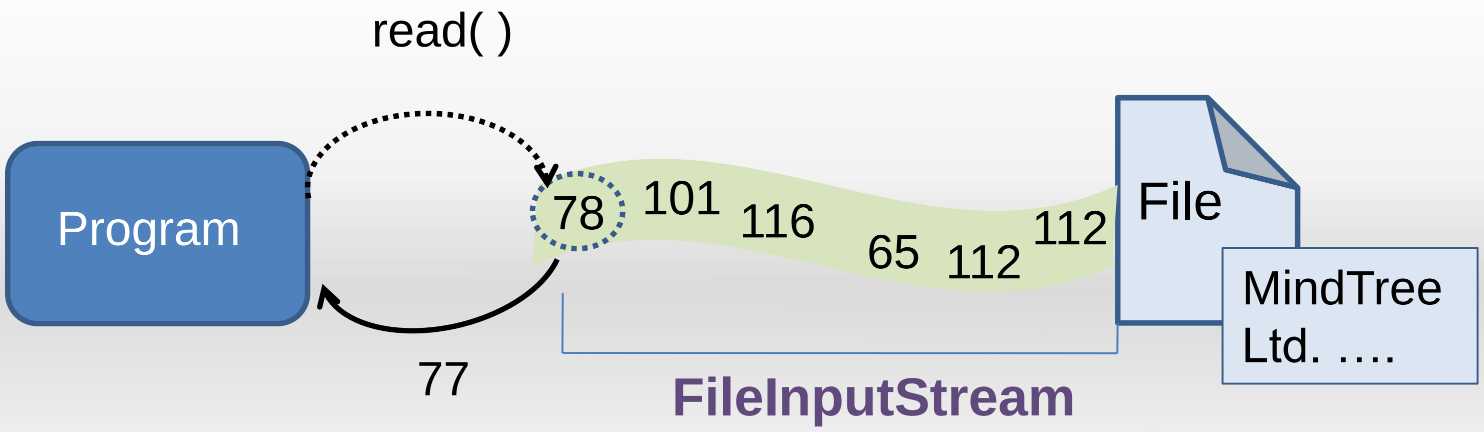
InputStream class methods

- read() reads a single byte. It returns the number of bytes read, or -1 if end-of-file has been reached.
- read(byte[]), which takes the byte array, reads an array of bytes and returns a -1 if the end-of-file has been reached.
- skip(long), skips a specified number of bytes of input and returns the number of bytes actually skipped.
- available() returns the number of bytes that can be read without blocking.
- close() closes the input stream to free up system resources

InputStream
+ int : read () + int : read (byte[ ] data) + long: skip (long lng) + int: available ( ) + void : close ( )

FileInputStream

- A FileInputStream obtains input bytes from a file in a file system.
- FileInputStream is meant for reading streams of raw bytes such as image data.
- If you are trying to open the file which does not exist, or if you are opening a directory rather than a regular file, or for some other reason cannot be opened for reading FileInputStream throws FileNotFoundException.



Code Snippet: using FileInputStream to read text file.

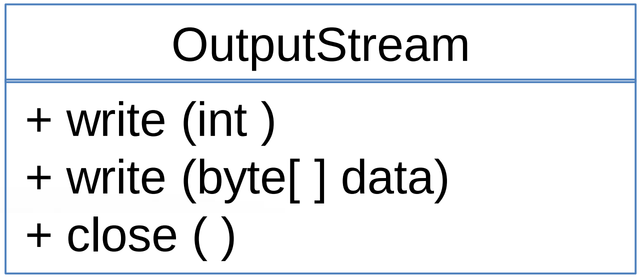
```
public static void main(String[] args) {
    FileInputStream fin = null;
    int b = -1;
    try {
        fin = new FileInputStream( "FILE NAME" );
        /* read() returns the next byte of data,
         * or -1 if the end of the file is reached.
         */
        while( (b = fin.read()) != -1) {
            System.out.print((char)b);
        }
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }finally {
        try {
            fin.close();
        } catch (IOException e) {
         System.out.println(e.getMessage());
        }
    }

}
```

Example

- Code Example:
  - Refer: FileInputStreamExample.java
    - Illustrates how to read byte by byte from a text file.

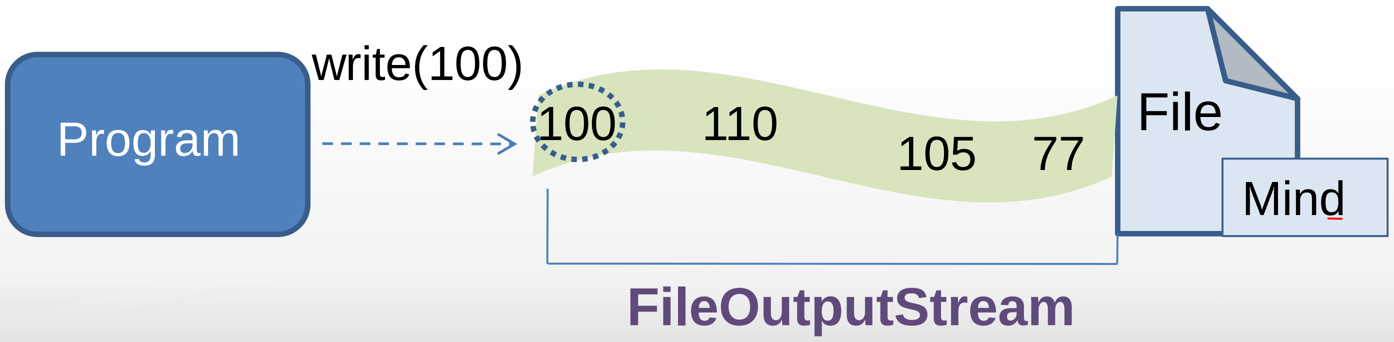
OutputStream class

- write(int b)
  Writes the specified byte to the output stream..
- write(byte[] b)
  Writes b.length bytes from the specified byte array to this output stream.
- close() closes the output stream to free up system resources



FileOutputStream

- A file output stream is an output stream for writing data to a File.
- FileOutputStream is meant for writing streams of raw bytes such as image data.



Using FileOutputStream to write data to file

```
String strData = new String("MindTree Ltd.");
byte[] data = strData.getBytes();
/*
 * Opens “FILE NAME”
 * If file exists it overwrites the contents.
 */
FileOutputStream fout = new FileOutputStream( “FILE NAME” );
fout.write(65); //write character 'A' to file.
fout.write(data); // write "MindTree Ltd." to a file
```

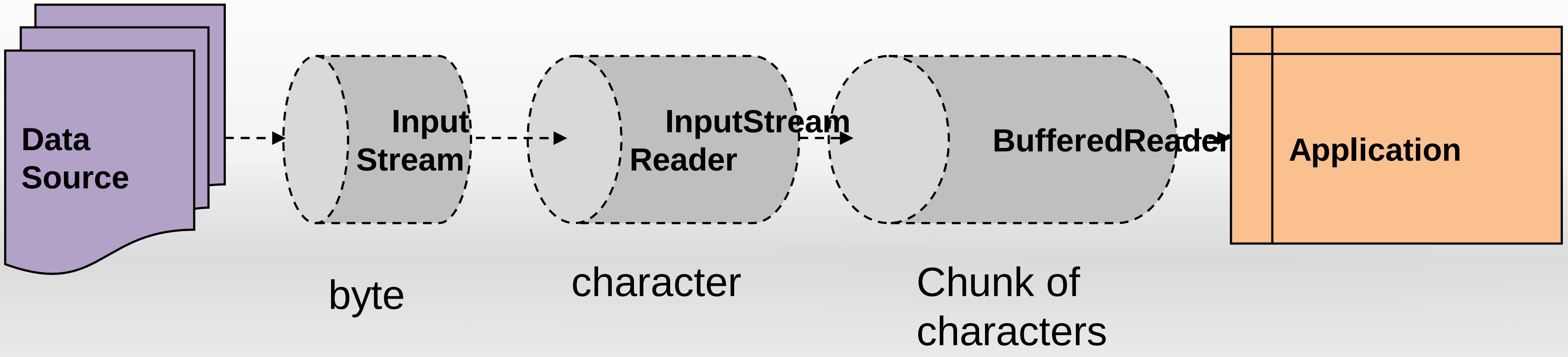
```
/*
 * Second argument of type boolean is for append mode
 * true --> append
 * false --> overwrite
 */
FileOutputStream fout = new FileOutputStream( “FILE NAME” ,true);
```

Example

- Code Example
  - Refer FileOutputStreamExample.java
    - Illustrates how to write to a file using byte stream

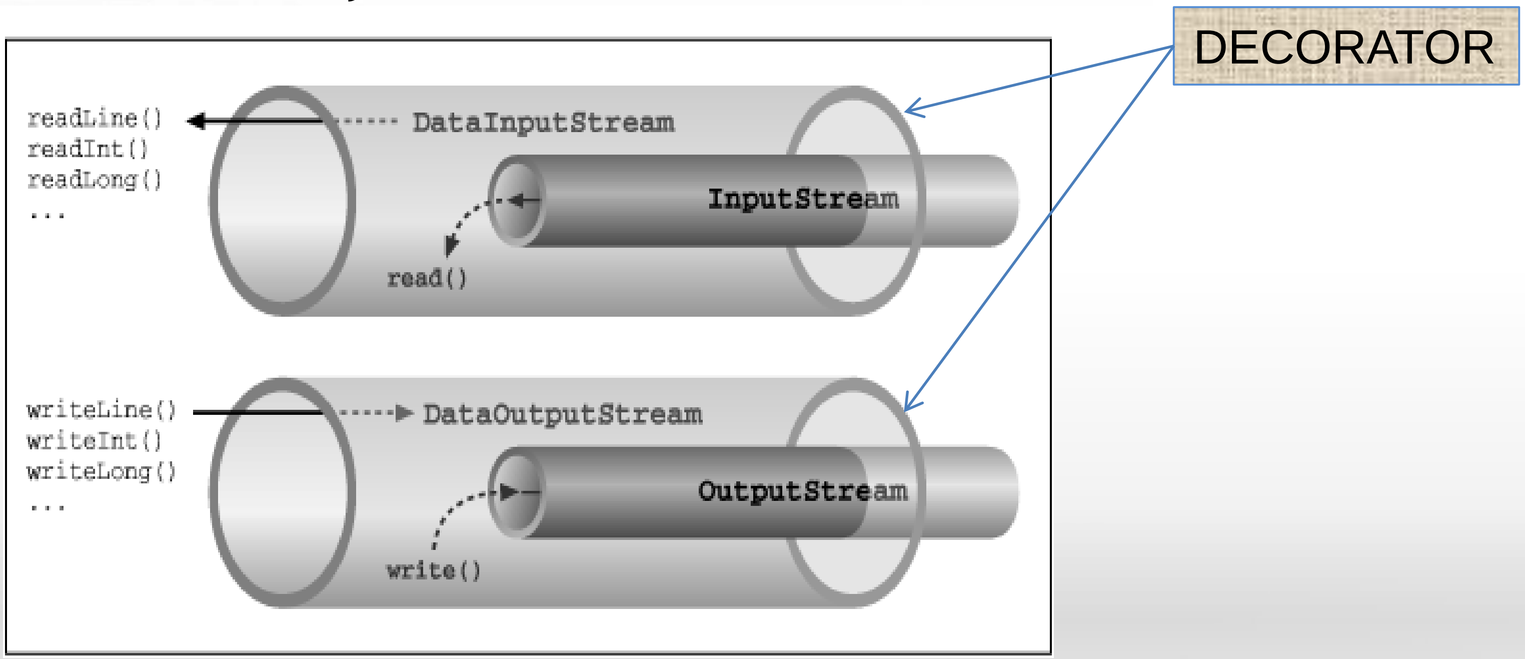
Wrapper classes

- Streams send or receive data between the application and the data source/destination in their most basic level.
- Requirement may be there in an application to read /write java primitive data types or read /write objects.
- Wrapper classes exist in order to provide methods to provide additional higher level functionality when dealing with streams.
- The various wrapper class are meant to be ‘chained’ together



FilterInputStream and FilterOutputStream class

- These streams sit on top of an already existing streams (the *underlying* stream) which it uses as its basic sink of data, but possibly transforming the data along the way or providing additional functionality.



FilterOutputStream classes.

- Some important sub classes of FilterOutputStream:
  - BufferedOutputStream
    The class implements a buffered output stream.
  - DataOutputStream
    A data output stream lets an application write primitive Java data types to an output stream in a portable way
  - PrintStream
    A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
  - ZipOutputStream
    This class implements an output stream filter for writing files in the ZIP file format

FilterInputStream classes.

- Some important sub classes of FilterInputStream:
  - BufferedInputStream

    A BufferedInputStream adds functionality to another input stream-namely, the ability to buffer the input.
  - DataInputStream

    A data input stream lets an application read primitive Java data types from an underlying input stream.
  - ZipInputStream

    This class implements an input stream filter for reading files in the ZIP file format.

Code Snippet: Using DataOutputStream class

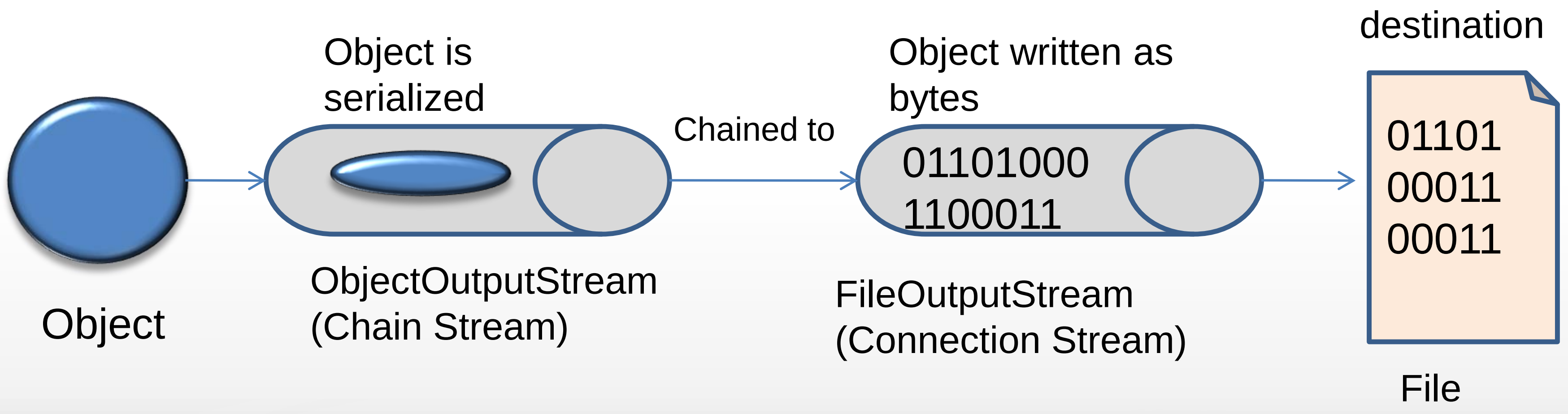
```
/*
 * Open a file for writing using
 * FileOutputStream. Using this stream you can only
 * write byte by byte.
 */
FileOutputStream fout = new FileOutputStream("d:\\test.dat");
 /*
  * You cannot use any Filter Streams on their own.
  * Decorate DataOutputStream on FileOutputStream,
  * you can write java primitive data types to a file.
  * Similarly if you decorate DataOutputStream on SocketStream's,
  * you can write java primitive data types to a network socket.
  */
 DataOutputStream dataOut = new DataOutputStream(fout);
 dataOut.writeInt(22); // 4 bytes
 dataOut.writeChar('J'); // 2 bytes
 dataOut.writeDouble(1.2); // 8 bytes
```

Code Snippet: Using DataInputStream class

```
/*
 * Open a file for reading using FileInputStream.
 * Using this stream you can read only bytes.
 */
FileInputStream fin = new FileInputStream("d:\\test.dat");

/*
 * Decorate DataInputStream on FileInputStream,
 * you can read java primitive data types from a file.
 * Similarly if you decorate DataInputStream on SocketStream's
 * you can read java primitive data types from a network socket.
 */
DataInputStream dataIn = new DataInputStream(fin);

System.out.println(dataIn.readInt()); // read 4 bytes
System.out.println(dataIn.readChar()); // read 2 bytes (Unicode)
System.out.println(dataIn.readDouble()); // read 8 bytes
```

Serialization

- Serialization allows instances of an object to be represented as a stream, which can then be written to a data source/destination.



Example

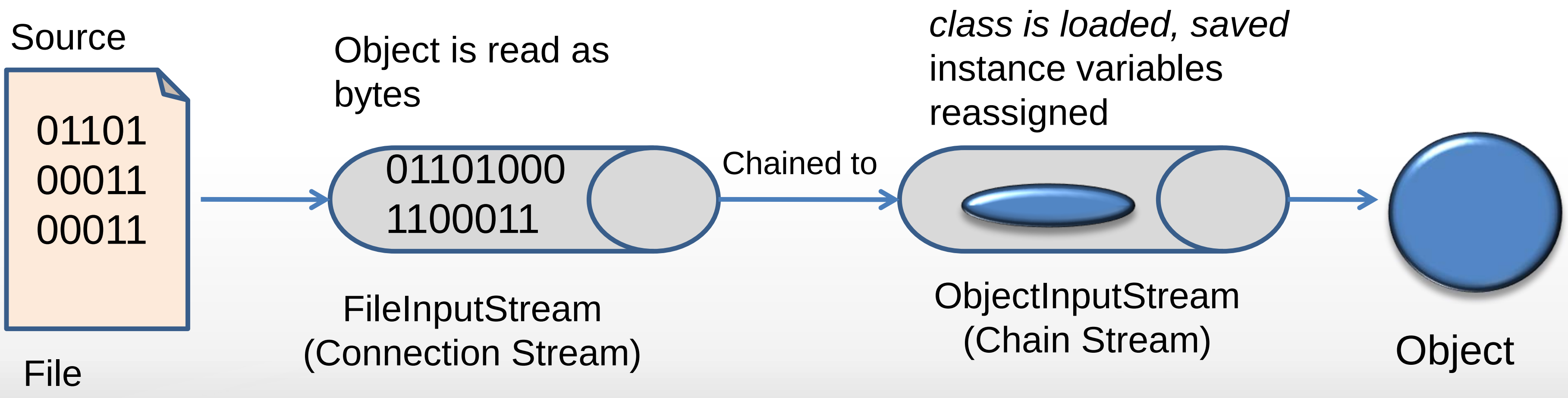
- Code Example:
  - Refer : DataStreamsExample.java
    - Code illustrates using reading and writing java primitive types using Wrapper Stream classes.

Code Snippet : Serialization.

```
/*
 * Open a file to write bytes
 */
FileOutputStream fout = new FileOutputStream("d:\\test.dat");

/*
 * Use ObjectOutputStream to serialize an object
 * to a file system
 */

ObjectOutputStream out = new ObjectOutputStream(fout);

/*
 * Create an instance of String and java.util.Date.
 * Both String and Date class implements Serializable interface.
 */
String strData = new String("MindTree Ltd.");
Date today = new Date();

out.writeObject(strData);    // write String to a file stream
out.writeObject(today);      // write date to a file stream.
```

Deserialization

- *Deserialization* is the process of rebuilding those bytes into a live object.



Code Snippet: Deserialization

```
/*
 * Open file for reading bytes.
 */

FileInputStream fin = new FileInputStream("d:\\test.dat");

/*
 * use ObjectInputStream to deserialize an object
 * coming from file system
 */

ObjectInputStream in = new ObjectInputStream(fin);

/*
 * read String
 */
String str = (String) in.readObject();
System.out.println(str);

/*
 * read Date
 */
Date date = (Date) in.readObject();
System.out.println(date);
```

Serialization

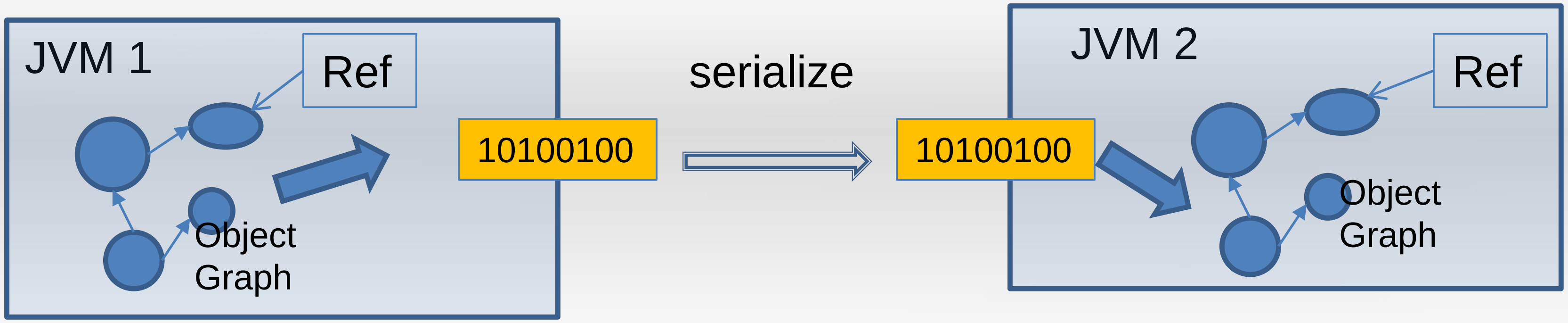
- What type of objects should be serialized ?
  - Only objects which are self contained should be serialized.
    - Example: Entity classes (Customer, Product, Book, …)
- What type of objects should not be serialized ?
  - Objects which are depending on some other external resources should not be serialized.
    - Example: any class containing reference to files/ database connections / socket connections
- How to mark a class as self contained.
  - Use a marker interface **java.io.Serializable**.

Note: Marker interfaces do not contain any methods. They are going to change the behavior of a class

Using Serialization

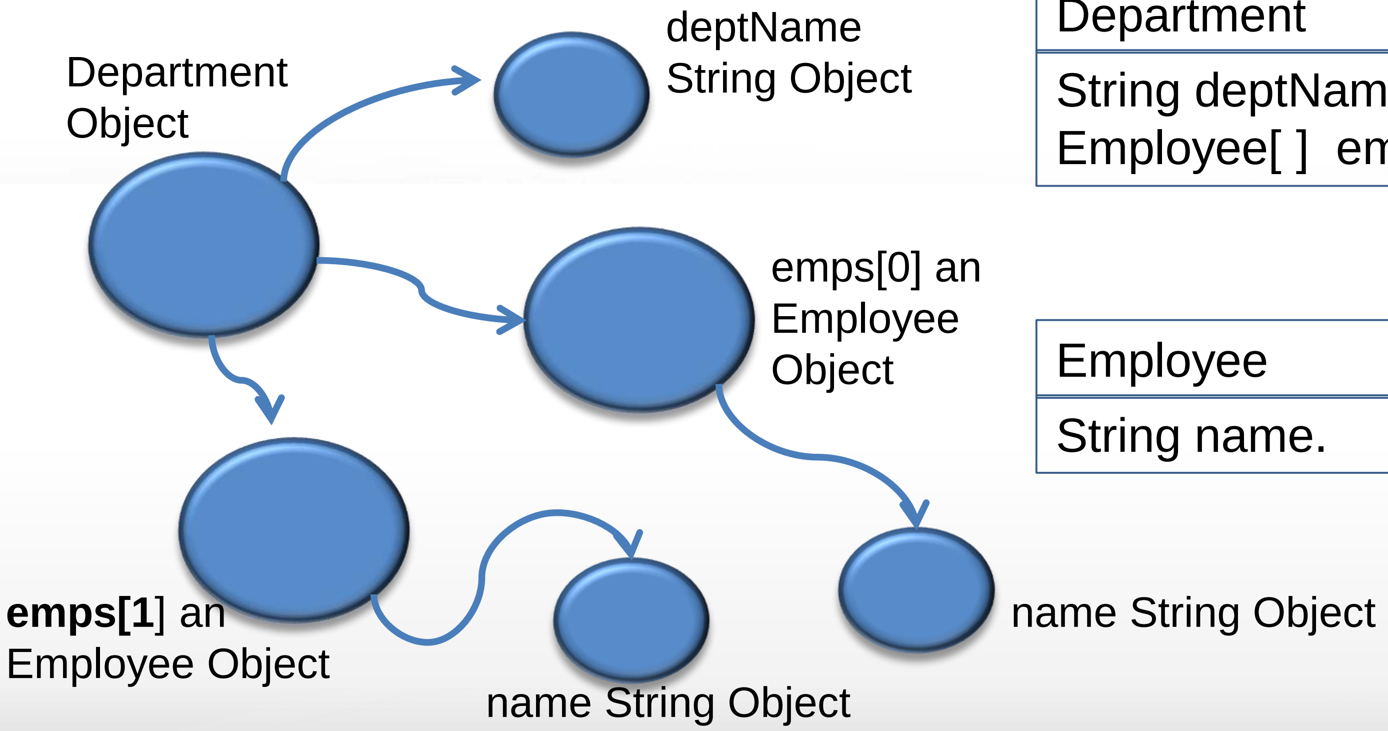
Three main uses of serialization:

1. As a persistence mechanism if the stream being used is FileOutputStream, then the data will automatically be written to a file.
2. As a copy mechanism if the stream being used is ByteArrayOutputStream, then the data will be written to a byte array in memory. This byte array can then be used to create duplicates of the original objects.
3. As a solution to implement call-by-value semantics in Distributed Computing using sockets.



Serialization and Object Graph

- Serialization saves the entire object graph.



Department
String deptName Employee[ ] emps.

Employee
String name.

When you save an object of type Department, it serializes deptName an instance of String and also all the employee instances. Each member of employee instance also gets serialized.

Note: Department and Employee should implement java.io.Serializable interface.

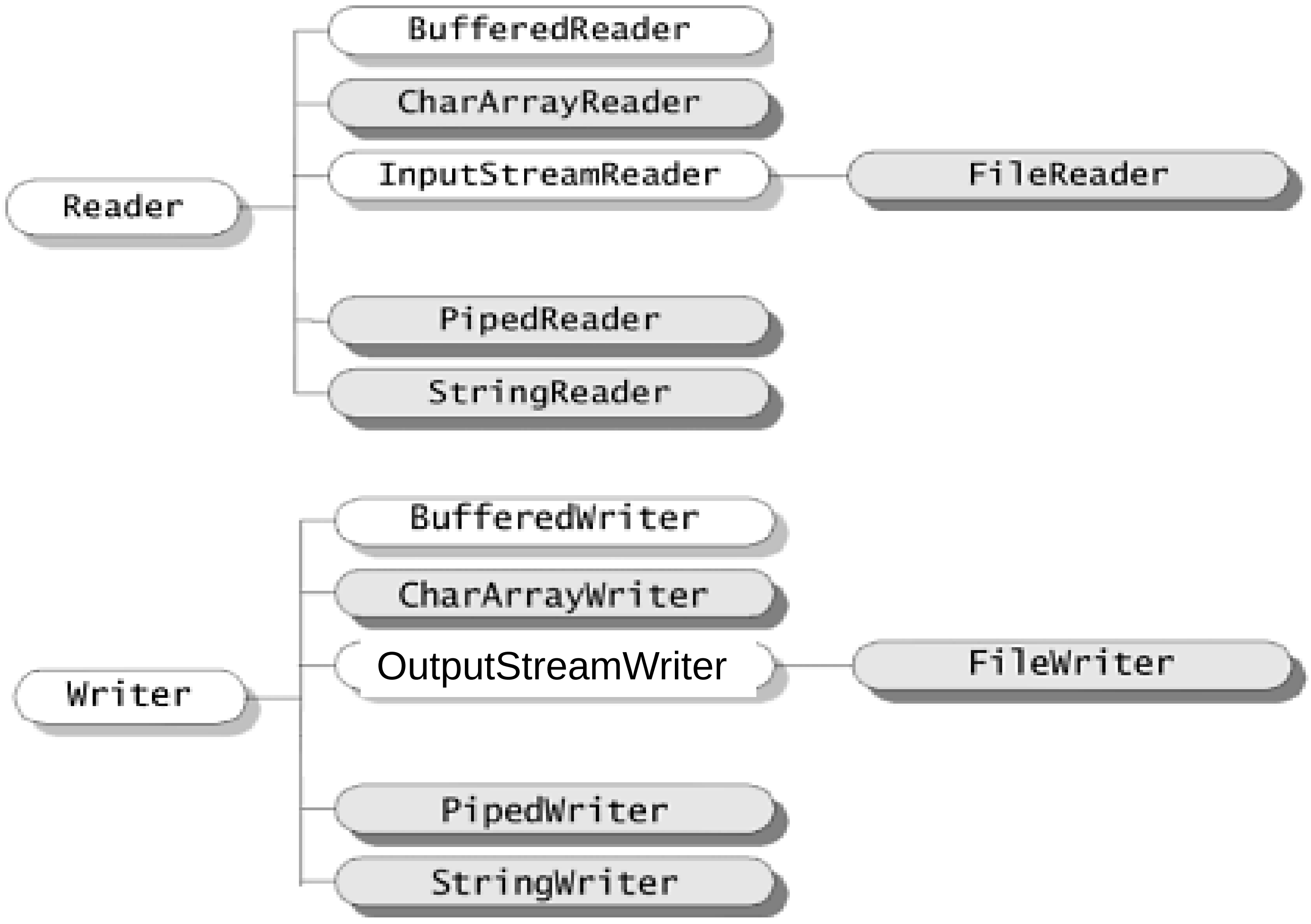
Example:

- Code Example:
  - Refer: SerializationExample.java
  - Illustrates how to serialization and deserialization
- Video:
  - Serialization.swf
    - Illustrates how to serialize an user defined entity class.

Character Streams

- Character streams that are subclasses of **Reader** read 16-bit Unicode characters
- Character streams that are subclasses of **Writer** write 16-bit Unicode characters

Character Stream Hierarchy



Reading from Keyboard.

```
/*  System.in: InputStream which can read bytes
 *  InputStreamReader is a bridge class which converts
 *  byte stream to character stream.
 *  Using InputStreamReader you can read characters
 */
InputStreamReader reader  = new InputStreamReader(System.in);

/*
 * BufferedReader can be used to read lines.
 *
 */
BufferedReader keyBoard = new BufferedReader(reader);
System.out.println("Enter Name:");
/*
 *  readLine() method of BufferedReader
 *  returns a String read from KB.
 */
String name = keyBoard.readLine();
```

Quiz!

Some quiz questions to reinforce the classroom learning

Quiz Questions

- What happens when you execute the following statement ?
  FileOutputStream fout = new FileOutputStream(“a.txt”);
  - Creates a file a.txt in append mode
  - Create file a.txt if file does not exist and overwrites if it exist
  - Create file a.txt if file does not exist, if file exists throws exception
  - compiler error: no constructor found.

- Which keyword used to specify that the instance variable should not be Serialized?

- Why is it easier to save an object with an ObjectOutputStream than a BufferedWriter or DataOutputStream?

Quiz questions

- What is the result of serializing an Object of Student class?

```
class Person  implements java.io.Serializable  {
  private String name;
  private static String place;
     // remaining code
}
class Student extends Person{
  private double marks;
     // remaining code
}
```

Quiz questions

- What is the result of serializing an Object of **Book** class?

```
class Publisher {
      private String name;
      // remaining code
}
class Book implements java.io.Serializable {
      private String title;
      private Publisher publisher;
      private double price;
     // remaining code
}
```

Explore More!!

Never let your curiosity die!

Explore more

- Character Streams
- Decorator design pattern
- Using Externalizable instead of Serializable interface to serialize.
- How to use Channels and buffers on Java NIO
- Using PipedInputStream and PipedOutputStream for inter thread communication.
- How to read and write compressed data using ZipOutputStream and ZipInputStream of java.util.zip package
- Apache Commons I/O - http://goo.gl/SJmBF

References

Contains the reference that will supplement the self learning and will be needed for completing the assignments & practice questions

References

- Java tutorials : Basic I/O
  - http://goo.gl/aCDcP
- Java I/O code examples
  - **http://goo.gl/QUYEZ**
- Java Serialization
  - http://goo.gl/gJCAi
  - http://goo.gl/Ua5Gk
- Java NIO
  - http://goo.gl/8uC49