java.io

Shristi Technology Labs

Contents

- Overview
- Streams
- ByteStreams
- CharacterStreams
- Serialization

Overview

- Java io package provides classes for system input and output through data streams, serialization and the file system.

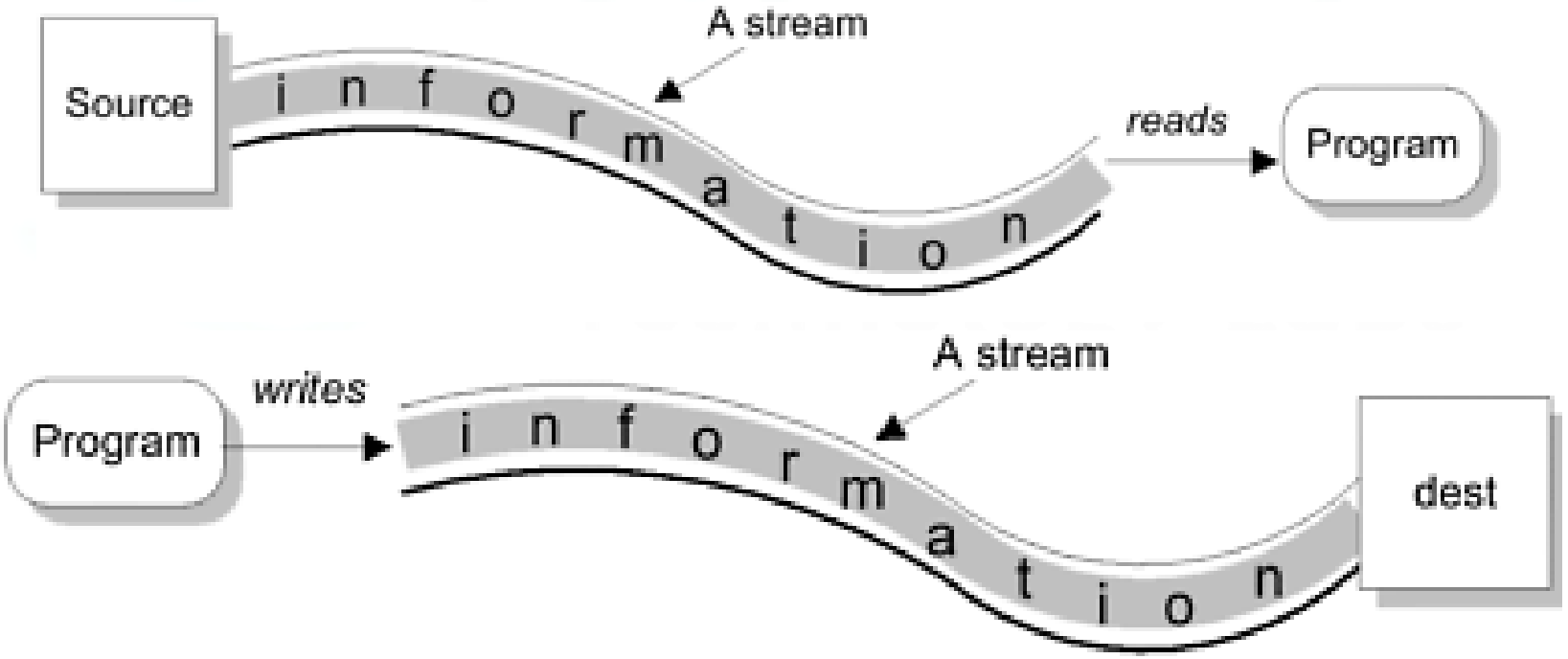
Streams

- Byte Streams
- Character Streams
- Pre-defined Streams

Stream

I/O in Java is based on streams.

A stream represents a flow of data, or a channel of communication with a writer at one end and a reader at the other.



Streams in Java IO

InputStream & OutputStream

- Abstract classes that define the basic functionality for reading or writing an **sequence of bytes**.
- All other byte streams in Java are built on top of the basic InputStream and OutputStream.

Reader & Writer

- Abstract classes that define the basic functionality for reading or writing an **sequence of characters**.
- All other character streams in Java are built on top of Reader and Writer.

Streams in Java IO

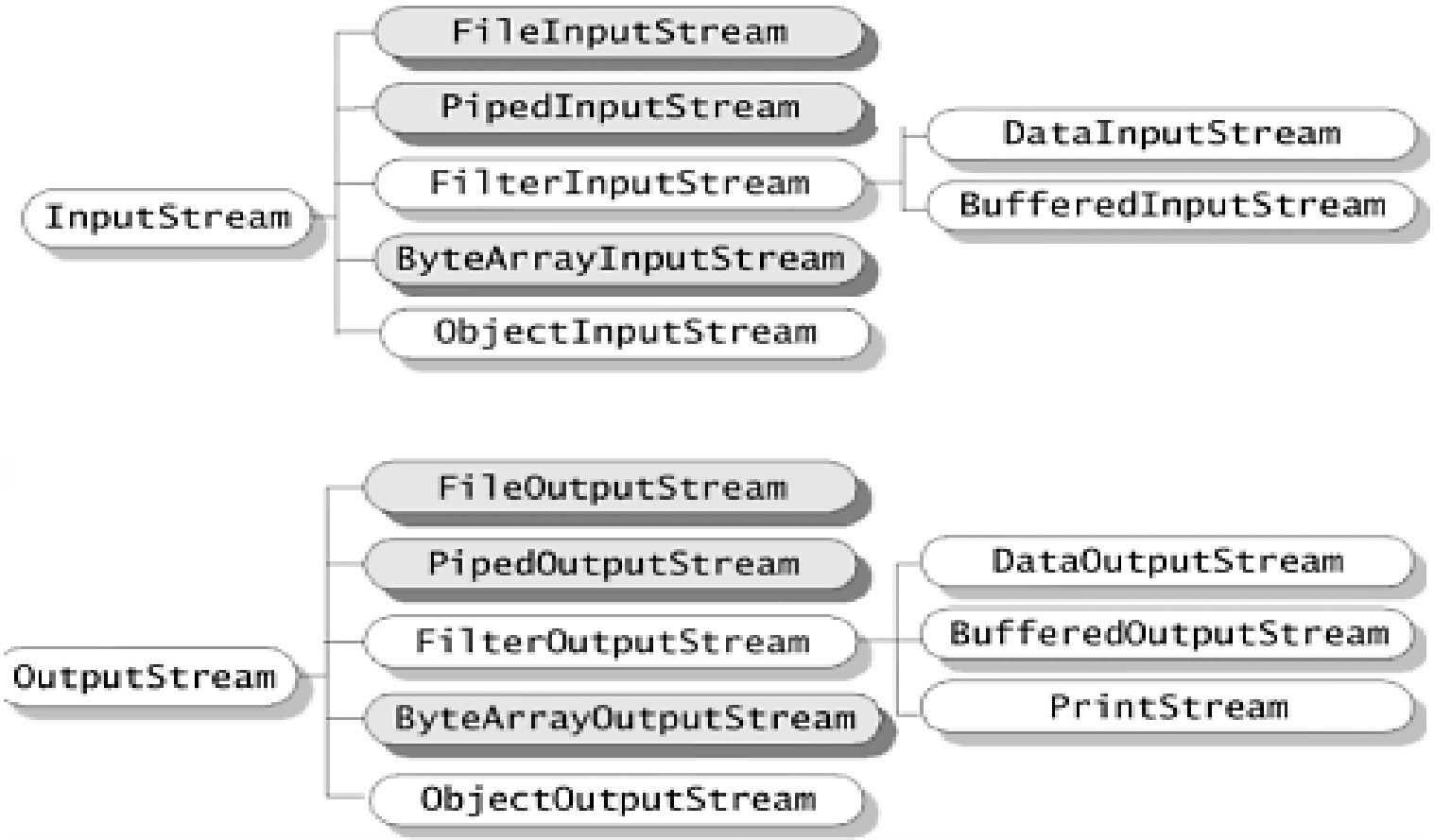
InputStreamReader/OutputStreamWriter

- The "Bridge" classes that convert bytes to characters and vice versa.

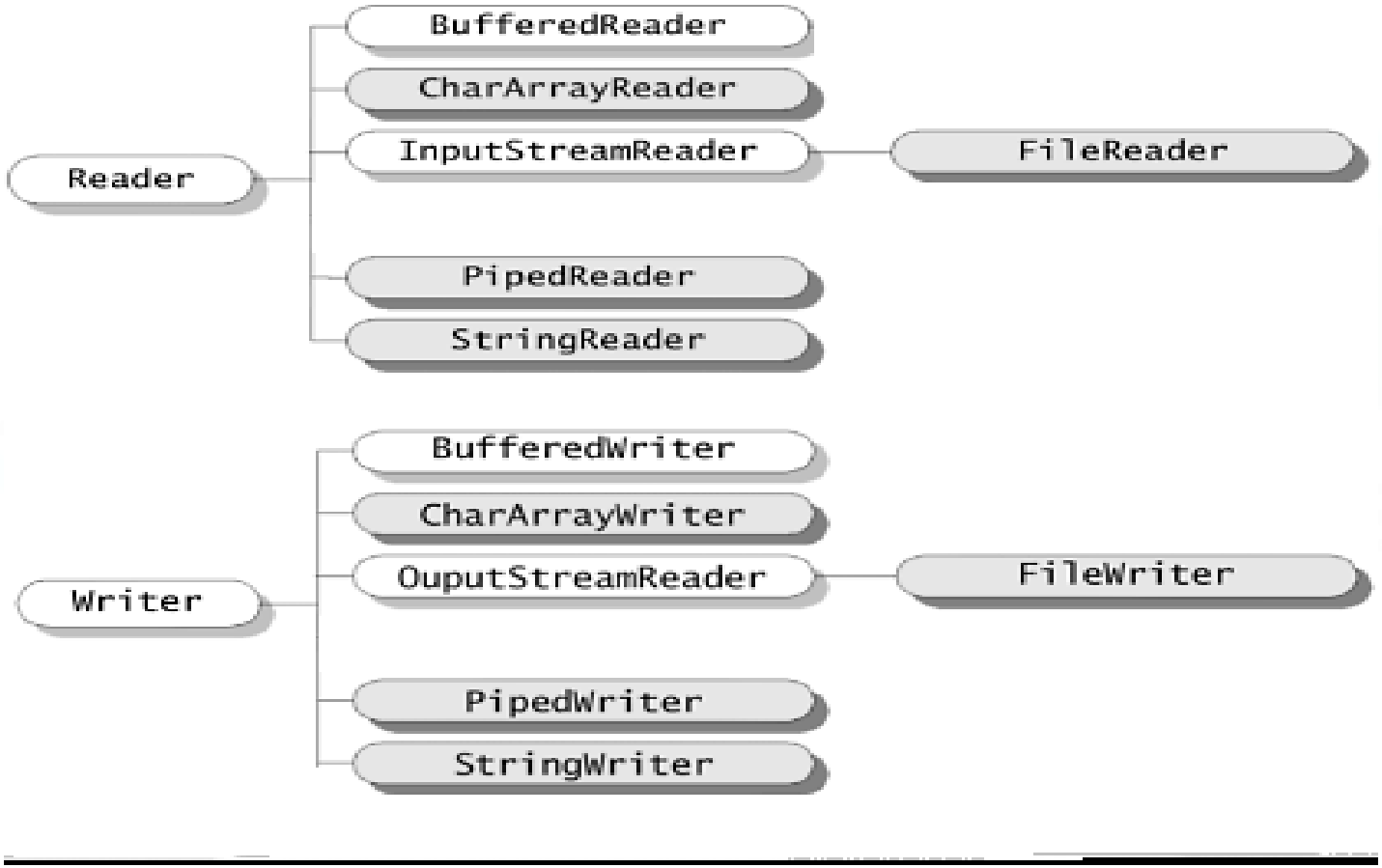
DataInputStream/DataOutputStream

- Specialized stream filters that add the ability to read and write simple data types like numeric primitives and String objects.

Byte Streams



Character Streams



BufferedReader

- Read text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.
- The buffer size may be specified, or the default size may be used.
- The default is large enough for most purposes.
- Use the constructor **BufferedReader(Reader obj)** to create a BufferedReader object

Working with BufferedReader

BufferedReader(Reader obj)

- Takes Object of Reader as parameter in constructor
- Reader is an abstract class, use **InputStreamReader** a sub class of Reader

InputStreamReader(InputStream obj)

- subclass of Reader – converts bytes to characters
- Takes object of InputStream as parameter in constructor
- **System.in** is an object of inputStream

Methods

Method of InputStream

- abstract int read() – reads the next byte of data from the input stream

Method of OutputStream

- abstract void write(int b) – Writes the specified byte to this output stream.

Method of BufferedReader

- readLine() – used to read a line of text

Serialization

- Is used to save the state of the object
- Can stream an object to a sequence of bytes and restore these objects from this stream of bytes.
- Implement **Serializable** interface to make an object serializable
- Serializable is a marker interface
- **ObjectInputStream** and **ObjectOutputStream** are the streams having the methods for serializing and deserializing an object.

transient

- Is a keyword used with instance variables
- Uses when a particular variable value should not be serialized
- Returns the default value of the data type on deserializing

Syntax:

```
transient String hobby;  //returns null on deserializing
```

From Java Spec

Variables may be marked transient to indicate that they are not part of the persistent state of an object

Classes for Serialization

ObjectOutputStream

- Used to serialize an object

Method

public final void writeObject(Object x) throws IOException

ObjectInputStream

- Used to deserialize an object

Method

public final Object readObject() throws IOException

Process of Serialization

```
Student student = new Student();
student.setName("Ram");
student.setDepartment("admin");

FileOutputStream fs = new FileOutputStream("test.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(student);
```

Process of DeSerialization

```
FileInputStream fs = new FileInputStream("test.ser");
ObjectInputStream os = new ObjectInputStream(fs);

Student student = (student)os.readObject();
System.out.print(student.getName());
System.out.print(student.getDepartment());
```

Example

- Create a **Student** class with instance variables and getter /setter methods
- **Student** class must implement Serializable
- Create a **SerialStud** class to serialize the student Object
- **test.ser** file gets generated
- Create a **DeSerialStud** class to read the test.ser and deserialize to get the object back

Student.java

```
public class Student implements Serializable {

    private static final long serialVersionUID = 1L;

    String name;    int studid;
    String department;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getStudid() {
        return studid;
    }
    public void setStudid(int studid) {
        this.studid = studid;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }

}
```

SerialStud.java

```
public class SerialStud {

   public static void main(String[] args) {

        Student s = new Student();
        s.setName("Ramana");
        s.setStudid(10);s.setDepartment("admin");
        FileOutputStream fs = null;
        ObjectOutputStream os = null;

            try {
                fs = new FileOutputStream("test.ser");
                os = new ObjectOutputStream(fs);
                os.writeObject(s);
            } catch (IOException e) {
                    e.printStackTrace();
            }finally{
                try {
                    os.close();
                    fs.close();
                } catch (IOException e) {

                    e.printStackTrace();
                }

            }   } }
```

DeSerialStud.java

```
public class DeserialStud {

   public static void main(String[] args) {
       FileInputStream fs = null;
       ObjectInputStream os = null;
       try {
           fs = new FileInputStream("test.ser");
           os = new ObjectInputStream(fs);
           Student st = (Student) os.readObject();
           System.out.println(st.getName()+" "+st.getDepartment());

       } catch (FileNotFoundException e) {
           e.printStackTrace();
       } catch (IOException e) {
           e.printStackTrace();
       } catch (ClassNotFoundException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
       }finally{
           try {
               os.close();
               fs.close();
           } catch (Exception e) {
               e.printStackTrace();
           }
       }}  }
```

Thank You