I/O and scannar

I/O operations

Three steps:

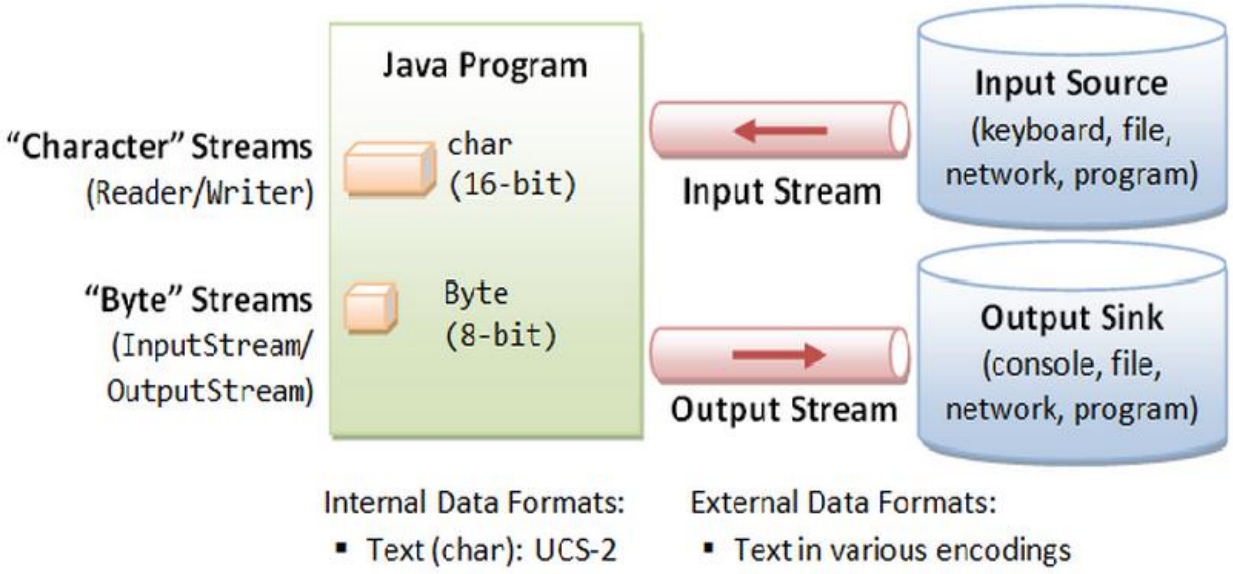
- *Open* an input/output stream associated with a physical device (e.g., file, network, console/keyboard), by constructing an appropriate I/O stream instance.
- *Read* from the opened input stream until "end-of-stream" encountered, or *write* to the opened output stream (and optionally flush the buffered output).
- *Close* the input/output stream.

Basic I/O - Stream

- Programs read inputs from data sources (e.g., keyboard, file, network, memory buffer, or another program) and write outputs to data sinks (e.g., display console, file, network, memory buffer, or another program). In Java standard I/O, inputs and outputs are handled by the so-called ***streams*****.**
- A stream can be defined as a sequence of data.
- **Packages**
  - **Java.io (stream based IO)**
  - **Java.nio(Buffer and channel-Based IO)**

Stream Types

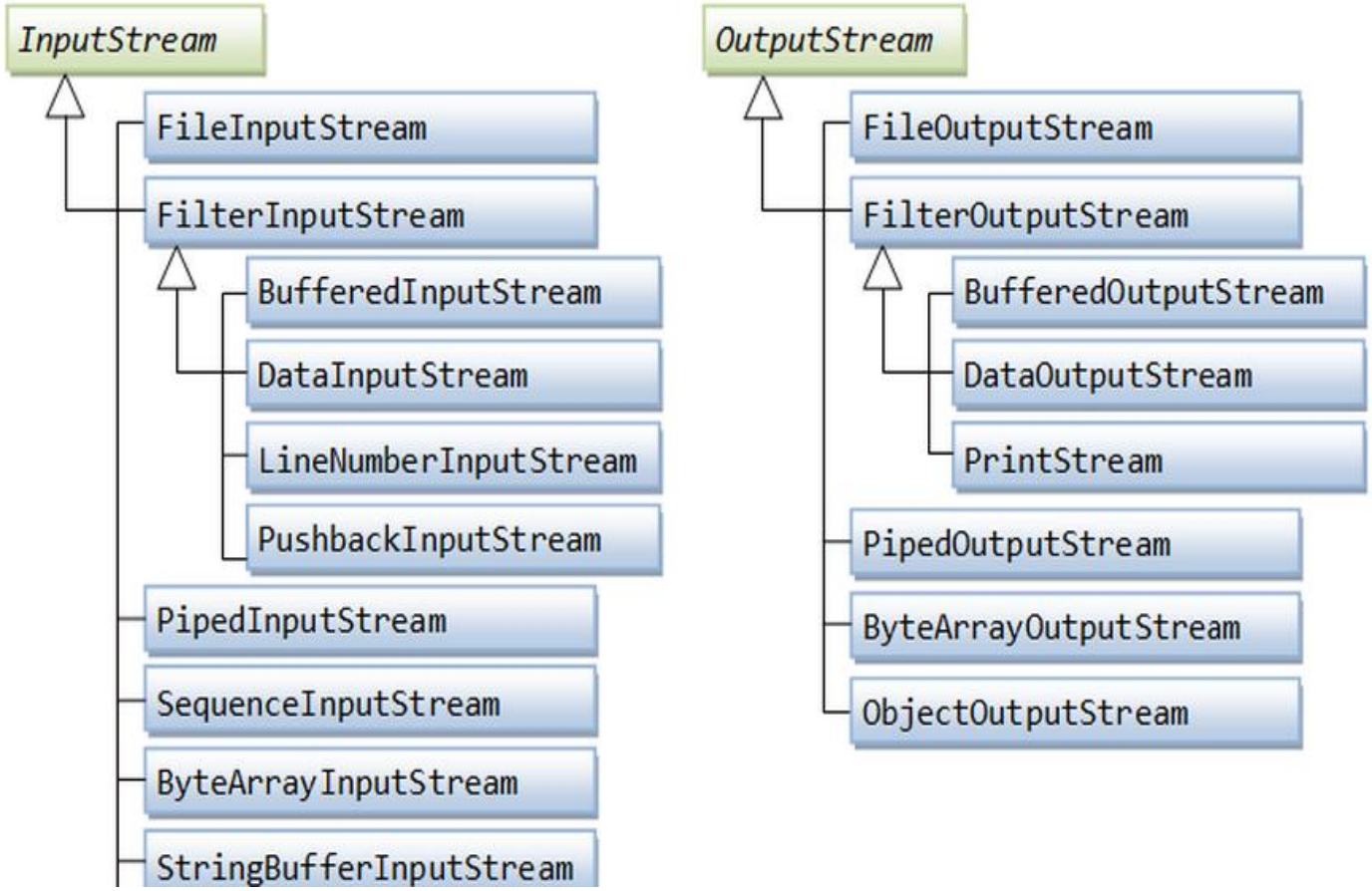
- Byte Streams
- Character Streams



Byte Streams – Abstract Class

- Programs use *byte streams* to perform input and output of 8-bit bytes.
- All byte stream classes are descended from InputStream and OutputStream.
- InputStream and OutputStream are abstract classes that cannot be instantiated
- Important methods:
  - read()
  - write()

Implementations of abstract InputStream/OutputStream



FileInputStream: Constructors

- **FileInputStream**(String name) Creates a FileInputStream by opening a connection to an actual file, the file named by the path name in the file system.
- **FileInputStream**(File file) Creates a FileInputStream by opening a connection to an actual file, the file named by the File object file in the file system.

InputStream: read methods`

- int read() :To read next one byte of data from the input source. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.
- int read(byte[] b): Reads some number of bytes from the input stream and stores them into the buffer array b. The number of bytes actually read is returned as an integer
- int read(byte[] b, int off, int len): Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer.
- All these methods throws IOException

FileInputStream: Constructors

- **FileOutputStream**(File file) Creates a file output stream to write to the file represented by the specified File object.
- **FileOutputStream**(File file, boolean append) Creates a file output stream to write to the file represented by the specified File object.
- **FileOutputStream**(String name) Creates a file output stream to write to the file with the specified name.
- **FileOutputStream**(String name, boolean append) Creates a file output stream to write to the file with the specified name.

If the second argument is true, then bytes will be written to the end of the file rather than the beginning.

OutputStream: write methods

- void write(int *unsignedByte*)
  - Writes the specified byte to this output stream.
- void write(byte[] *bytes*, int *offset*, int *length*)
  - Write "*length*" number of bytes, from the *bytes* array starting from offset of *index*.
- void write(byte[] *bytes*)
  - Write *length number of bytes, from the bytes array starting from offset of index*

Opening & Closing I/O Streams

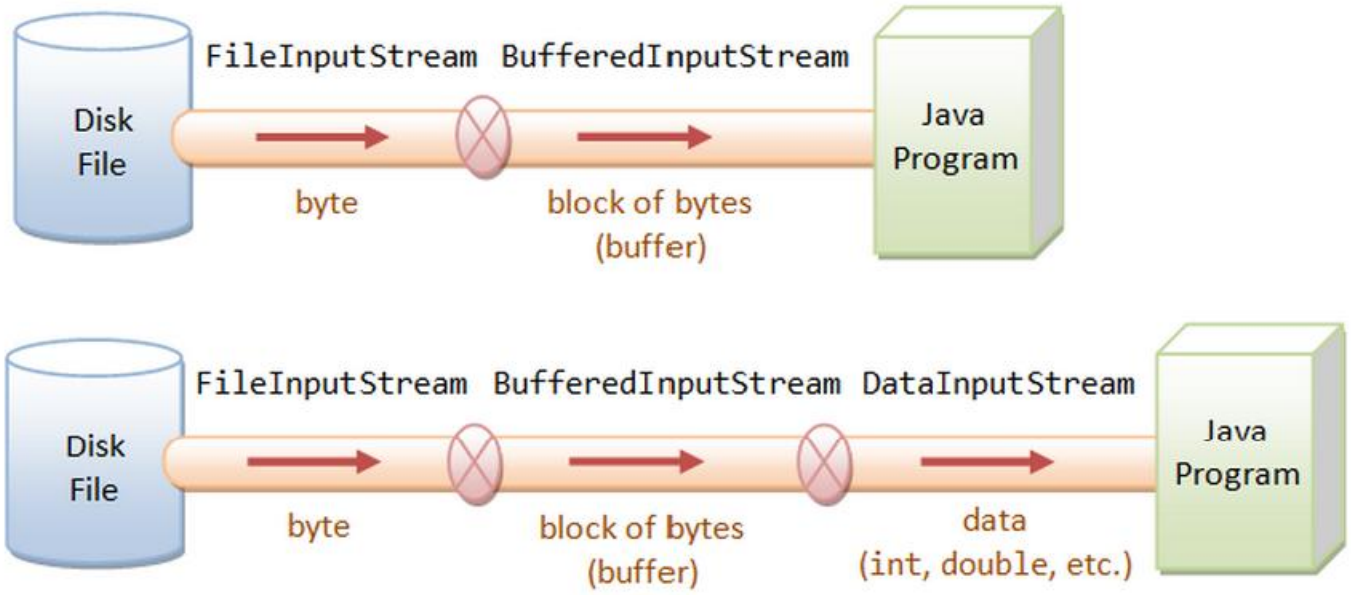
- public void **close**() throws IOException // close this Stream

```
FileInputStream in = null;
try { in = new FileInputStream(...); // Open stream
    ..... ..... }
catch (IOException ex) { ex.printStackTrace(); }
finally { // always close the I/O streams
 try { if (in != null) in.close(); } catch (IOException ex)
   { ex.printStackTrace(); } }
```

Flushing the OutputStream

- public void **flush**() throws IOException // Flush the output

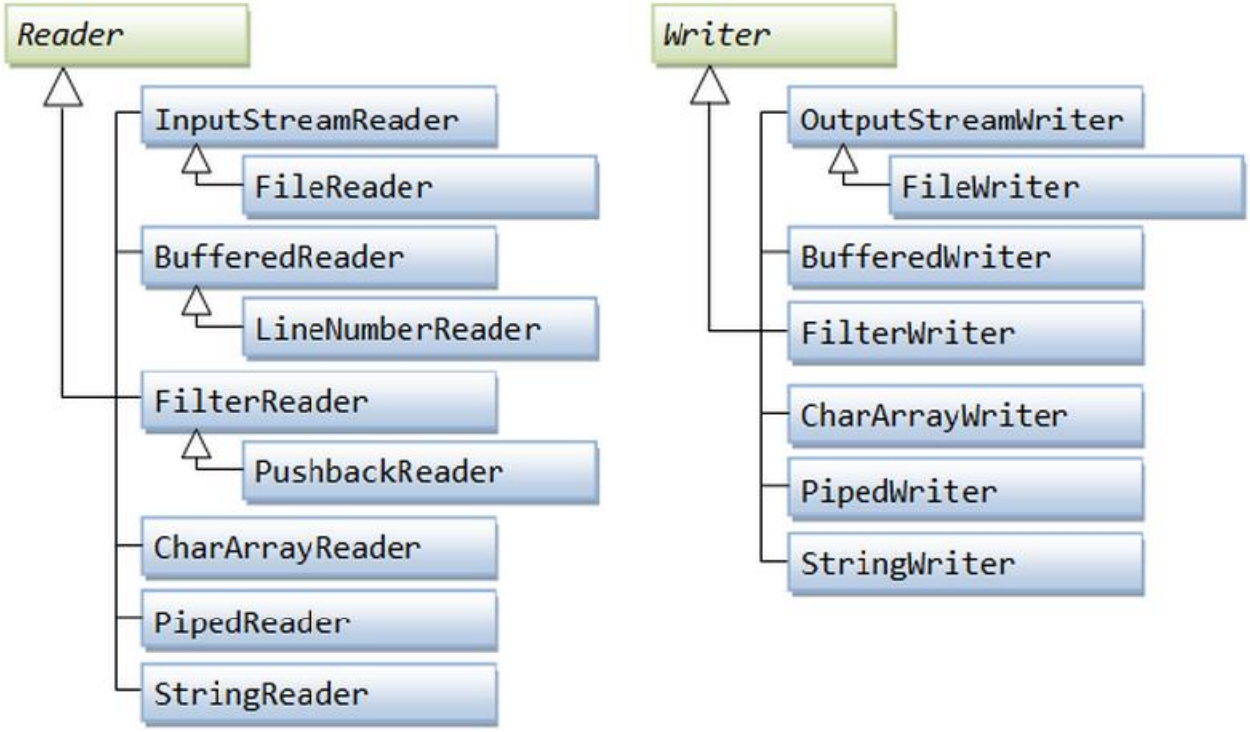
Layered (or Chained) I/O Streams



Character Streams – Abstract Class

- **Reader** is used to read data from a source
- **Writer** is used for writing data to a destination
- Important methods:
  - read()
  - write()

Character-Based I/O & Character Streams



Reader : Read method

- There are also variations of read() to read a block of characters into char-array.
- public abstract int **read**() throws IOException
- public int **read**(char[] *chars*, int *offset*, int *length*) throws IOException
- public int **read**(char[] *chars*) throws IOException

Writer: Write Method

- public void abstract void **write**(int *aChar*) throws IOException
- public void **write**(char[] *chars*, int *offset*, int *length*) throws IOException
- public void **write**(char[] *chars*) throws IOException

File I/O Character-Streams - FileReader & FileWriter

- FileReader and FileWriter are concrete implementations to the abstract super classes Reader and Writer, to support I/O from disk files.
- FileReader/FileWriter assumes that the *default character encoding* (*charset*) is used for the disk file. The default char set is kept in the JVM's system property

Buffered I/O Streams -

- **BufferedInputStream & BufferedOutputStream**

  –These stream can be stacked on top of (FileInputStream/FileOutputStream) to perform buffered I/O, instead of byte-by-byte
- **BufferedReader and BufferedWriter**
  - These streams can be stacked on top of FileReader/FileWriter or other character streams to perform buffered I/O, instead of character-by-character.
  - BufferedReader provides a new method readLine(), which reads a line and returns a String (without the line delimiter). Lines could be delimited by "\n" (Unix), "\r\n" (Windows), or "\r" (Mac).

Data I/O Streams

- Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values
- **DataInputStream**
  - A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way
  - readBoolean(), readByte(), readInt(), readFloat()
- **DataOutputStream**
  - A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in
  - writeBoolean(), writeByte(), writeInt(), writeFloat()

Object Streams -

- Object streams support I/O of objects. Most, but not all, standard classes support serialization of their objects. Those that do implement the marker interface Serializable
- **ObjectInputStream**
  - An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.
  - readObject()
- **ObjectOutputStream**
  - An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.
  - Persistent storage of objects can be accomplished by using a file for the stream
  - writeObject()

Files Class

File: Overview

- The File class in the Java IO API gives you access to the underlying file system
- This class gives you access to the file and file system meta data
- (JDK7)
- The java.nio.file package and its related package, java.nio.file.attribute, provide comprehensive support for file I/O and for accessing the default file system.

File : Constructors

- public **File**(String *pathName*)
  - Creates a new File instance by converting the given pathname string into an abstract pathname.
- public **File**(String *parent*, String *child*)
  - Creates a new File instance from a parent pathname string and a child pathname string.
- public **File**(File *parent*, String *child*)
  - Creates a new File instance from a parent abstract pathname and a child pathname string.
- public **File**(URI *uri*)
  - Constructs a File instance by converting from the given file-URI "file://....

File Constructor: Exmple

- A file relative to the current working directory
  - File file = new File("in.txt")
- A file with absolute path
  - File file = new File("d:\\myproject\\java\\Hello.java");
- A directory
  - File dir = new File("c:\\temp")

File/Directory: Checking Properties

- Tests if this file/directory exists.
  - public boolean **exists**()
- Returns the length of this file.
  - public long **length**()
- Tests if this instance is a directory.
  - public boolean **isDirectory**()
- Tests if this instance is a normal file.
  - public boolean **isFile**()
- Tests if this file is readable.
  - public boolean **canRead**()

File/Directory: Checking Properties

- Tests if this file is writable.
  - public boolean **canWrite**()
- Deletes this file/directory.
  - public boolean **delete**()
- Deletes this file/directory when the program terminates.
  - public void **deleteOnExit**()
- Renames this file.
  - public boolean **renameTo**(File *dest*)
- Makes (Creates) this directory.
  - public boolean **mkdir**()

Directory: Listing

- List of files as arrays of strings
  - public String[] list()
- List of files as Files
  - public File[] listFiles()
- Renames this file
  - public boolean **renameTo**(File *dest*)
- Makes (Creates) this directory.
  - public boolean **mkdir**()

Serialization

- *Serialization is the process of writing the* state of an object to a byte stream
- Only an object that implements the **Serializable interface can be saved and** restored by the serialization facilities
- The **Serializable interface defines no** members. It is simply used to indicate that a class may be serialized
- If a class is serializable, all of its subclasses are also serializable.

Serialization: transient

- **The transient** keyword is used in serialization.
- Any data member declared as transient, will not be serialized

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 transient int age;//Now it will not be serialized
  }
```

Cloning

- The **clone( ) method generates** a duplicate copy of the object on which it is called
- Only classes that implement the **Cloneable interface can be** cloned
- The **Cloneable interface defines no** members. It is used to indicate that a class allows a bitwise copy of an object (that is, a *clone) to be made*
- When a clone is made, the constructor for the object being cloned is *not called. A clone*is simply an exact copy of the original.

SCANNER Class

Interactive programs

- We have written programs that print console output, but it is also possible to read input from the console
- The user types input into the console. We capture the input and use it in our program
- Such a program is called an interactive program

Input and System.in

- System.out

  An object with methods named println and print
- System.in

  not intended to be used directly

  We use a second object, from a class Scanner, to help us.

Constructing a Scanner object

- Scanner is in a package named java.util

  import java.util.*;
- Scanner name = new Scanner(System.in);
- Example:

  Scanner console = new Scanner(System.in);

Scanner methods

Method

- nextInt()
- nextDouble()
- next()
- nextLine()

Each method waits until the user presses Enter.