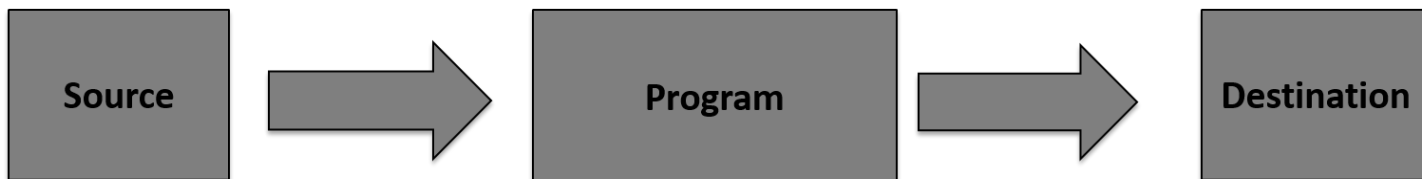


INPUT/OUTPUT

Tanjina Helaly

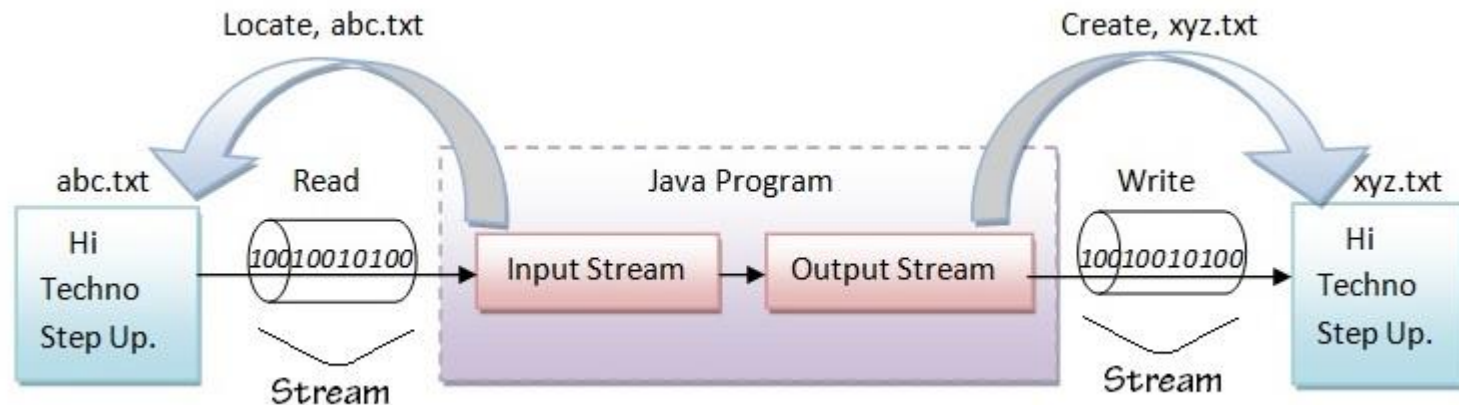
Input/Output in Java

- The java.io package - classes to perform input and output (I/O) in Java.
- All these streams represent
 - an input source and
 - an output destination.
 - supports many data such as primitives, Object, localized characters, etc.



Stream

- A stream can be defined as a sequence of data. there are two kinds of Streams
 - The Input Stream is used to read data from a source.
 - The Output Stream is used for writing data to a destination.



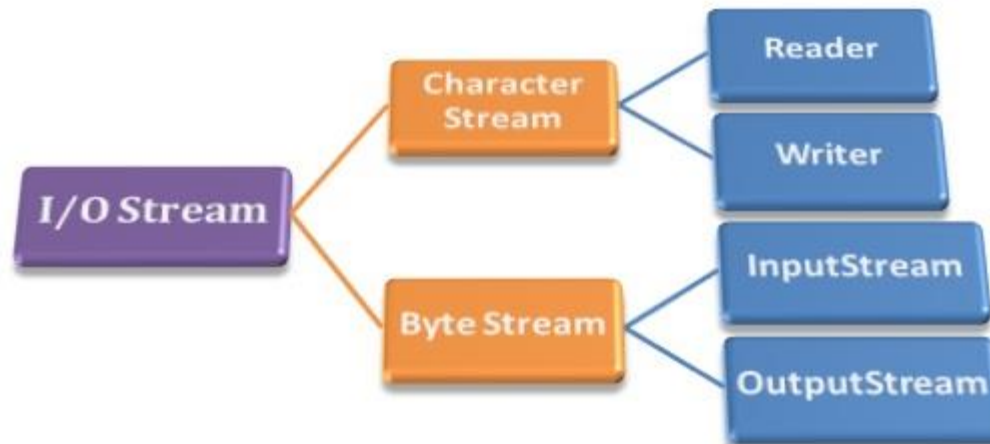
- Always close a stream when it's no longer needed
 - This practice helps avoid serious resource leaks.

I/O Stream Classifications

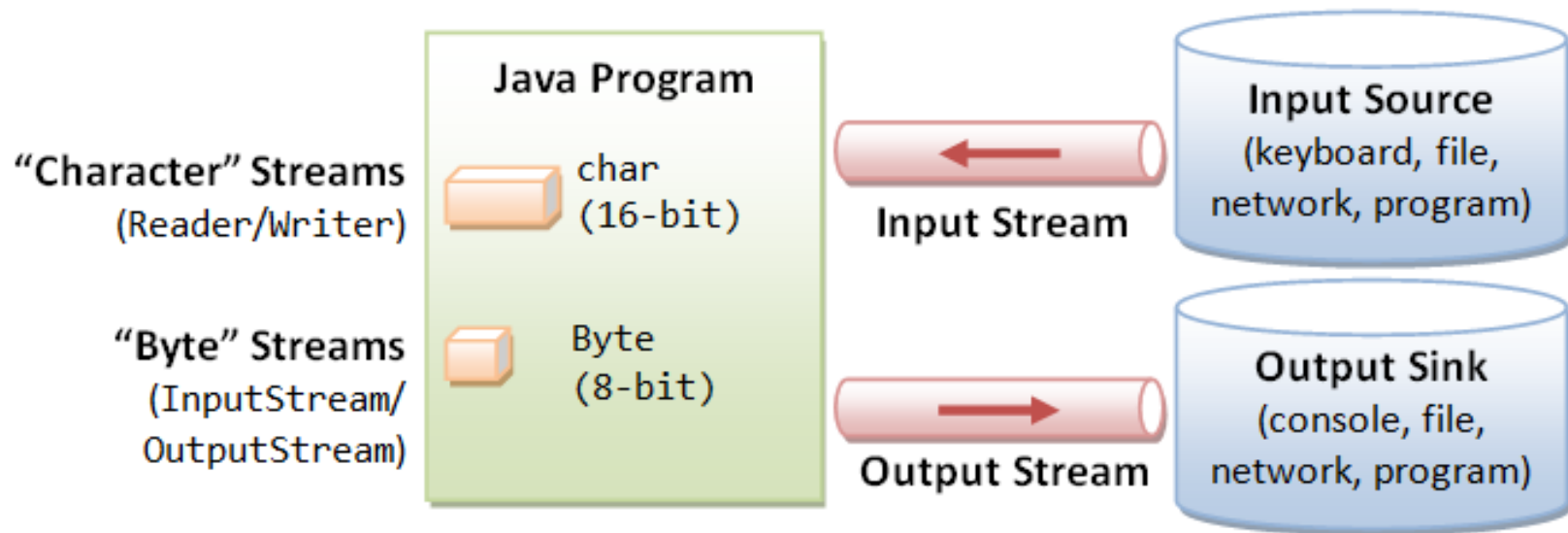
- [Byte Streams](#) handle I/O of raw binary data.
- [Character Streams](#) handle I/O of character data, automatically handling translation to and from the local character set.
- [Buffered Streams](#) optimize input and output by reducing the number of calls to the native API.
- [Scanning and Formatting](#) allows a program to read and write formatted text.
- [I/O from the Command Line](#) describes the Standard Streams and the Console object.
- [Data Streams](#) handle binary I/O of primitive data type and String values.
- [Object Streams](#) handle binary I/O of objects.

2 main I/O Stream

- Among the previously listed categories, two main category of IO classes are,
 - **Character- Oriented Stream:** It has two abstract classes **Reader** and **Writer**
 - **Byte- Oriented Stream:** It has two abstract classes **InputStream** and **OutputStream**



2 main I/O Stream



Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

Byte Format

- Programs use *byte streams* to perform input and output of 8-bit bytes.

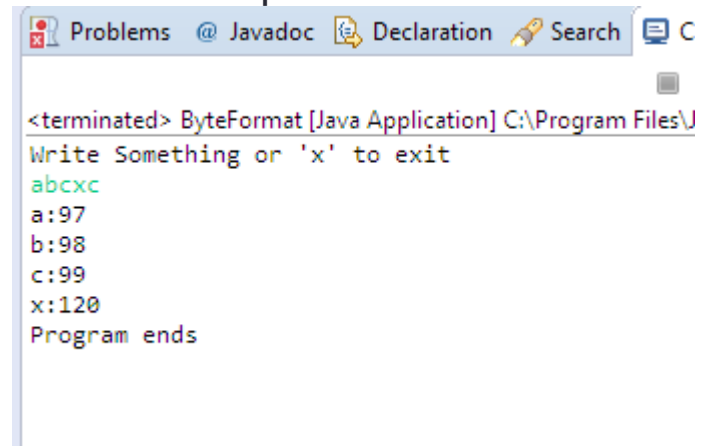
Base Class	Console	File	Methods
InputStream	System.in	FileInputStream(String) Or FileInputStream(File)	read() – return int (ASCII value of the character)
OutputStream	System.out(PrintStream)	FileOutputStream(String)) Or FileOutputStream(File)	write(int) write(byte[])

Byte Format – From Standard Input

```
import java.io.*;

public class ByteFormat {
    public static void main(String[] args) {
        InputStream is = System.in;
        int a=0;
        System.out.println("Write Something or 'x' to exit");
        while (a != 'x'){
            try {
                a = is.read();
                System.out.println((char)a + ":" + a);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        try { is.close();}
        catch (IOException e) { e.printStackTrace();}
        System.out.println("Program ends");
    }
}
```

Program Output

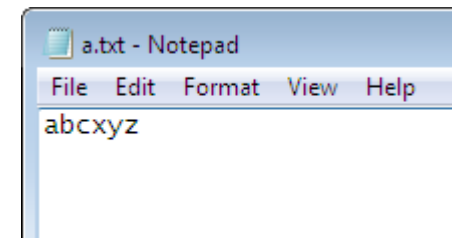


```
<terminated> ByteFormat [Java Application] C:\Program Files\J
Write Something or 'x' to exit
abcxc
a:97
b:98
c:99
x:120
Program ends
```


Byte Format – From File

```
import java.io.*;
public class ByteFormatFile {
    public static void main(String[] args) {
        int a=0;
        FileInputStream fis;
        try {
            fis = new FileInputStream("C:\\Temp\\a.txt");
            while((a = fis.read()) != -1)
                System.out.println((char)a + ":" + a);
            fis.close();
        }
        catch (FileNotFoundException e1) {
            e1.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Program ends");
    }
}
```

File Content



Program

Output

```
<terminated> ByteFormatFile [Java Application] C:\I
a:97
b:98
c:99
x:120
y:121
z:122
Program ends
```

Byte Format – Write to File

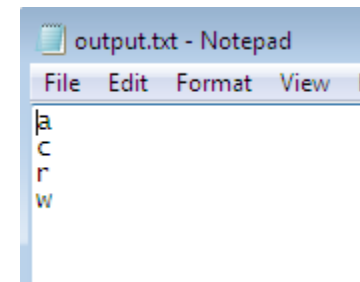
```
import java.io.*;

public class ByteFormatFile {
    public static void main(String[] args) {
        int a=0;
        FileOutputStream fos;
        InputStream is = System.in;
        try {
            fos = new FileOutputStream("C:\\Temp\\output.txt");
            System.out.println("Write Something or 'x' to exit");
            while ((a=is.read()) != 'x'){
                fos.write(a);
            }
            fos.close();
        }
        catch (FileNotFoundException e1) {
            e1.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Program ends");
    }
}
```

Program Output

```
<terminated> ByteFormatFile [Java Application] C:\Pro
Write Something or 'x' to exit
a
c
r
w
x
Program ends
```

File Content



PrintStream

- Derived from **OutputStream**
- The PrintStream class provides methods to write data to another stream(OutputStream, BufferedWriter, OutputStreamWriter).
- The PrintStream class automatically flushes the data so there is no need to call flush() method.
- Moreover, its methods don't throw IOException.

PrintStream

Constructor and Description

[PrintStream](#)([File](#) file) Creates a new print stream, without automatic line flushing, with the specified file.

[PrintStream](#)([File](#) file, [String](#) csn) Creates a new print stream, without automatic line flushing, with the specified file and charset.

[PrintStream](#)([OutputStream](#) out) Creates a new print stream.

[PrintStream](#)([OutputStream](#) out, boolean autoFlush) Creates a new print stream.

[PrintStream](#)([OutputStream](#) out, boolean autoFlush, [String](#) encoding) Creates a new print stream.

[PrintStream](#)([String](#) fileName) Creates a new print stream, without automatic line flushing, with the specified file name.

[PrintStream](#)([String](#) fileName, [String](#) csn) Creates a new print stream, without automatic line flushing, with the specified file name and charset.

PrintStream

<u>Modifier and Type</u>	<u>Method and Description</u>
PrintStream	append (char c) Appends the specified character to this output stream.
PrintStream	append (CharSequence csq) Appends the specified character sequence to this output stream.
PrintStream	append (CharSequence csq, int start, int end) Appends a subsequence of the specified character sequence to this output stream.
boolean	checkError () Flushes the stream and checks its error state.
protected void	clearError () Clears the internal error state of this stream.
void	close () Closes the stream.
void	flush () Flushes the stream.
void	print (boolean b) Prints a boolean value.
void	print (Object obj) Prints an object.
void	println (Object x) Prints an Object and then terminate the line.
void	println (String x) Prints a String and then terminate the line.
void	write (byte[] buf, int off, int len) Writes len bytes from the specified byte array starting at offset off to this stream.
void	write (int b) Writes the specified byte to this stream.

PrintStream

- Notes

- print/println() –

- has overloaded method for each primitive data, String and any Reference data.
 - For reference data print/println prints the output of the toString() method.
 - If the class doesn't override the toString() method, it will use the toString method of Object class which return the [classname@hashCode]

- write()

- works the same way as OutputStream
 - It takes the int input as the ASCII value and print the corresponding character

Character Format

- Handle I/O of character data, automatically handling translation to and from the local character set.

Base Class	Console	File	Method
Reader	InputStreamReader(System.in)	FileReader(String) or FileReader(File)	read() – return int (ASCII character of the character)
Writer	OutputStreamWriter(System.out)	FileWriter(String) Or FileWriter(File)	write(int) write(String) append(char)

Buffered Format

- For *unbuffered* I/O.
 - each read or write request is handled directly by the underlying OS.
 - since each such request often triggers
 - disk access,
 - network activity, or
 - some other operation that is relatively expensive.
- To reduce this kind of overhead – Buffered Stream
 - Buffered **input streams read** data from a memory area known as a *buffer*,
 - the native input API is called only when the buffer is **empty**.
 - Similarly, buffered **output streams write** data to a buffer,
 - the native output API is called only when the buffer is **full**.

Buffered Format

- There are four buffered stream classes used to wrap unbuffered streams:
 - [BufferedInputStream](#) and [BufferedOutputStream](#) create buffered byte streams,
 - [BufferedReader](#) and [BufferedWriter](#) create buffered character streams.
- **Flushing Buffered Streams**
 - Sometime we need to write out a buffer at critical points, without waiting for it to fill.

Buffered Byte Format

- Handle I/O of character data, automatically handling translation to and from the local character set.

Base Class	Console	File	Method
BufferedInputStream(InputStream)	BufferedInputStream(System.in)	BufferedInputStream(FileInputStream(String or File))	read() – return int (ASCII character of the character)
BufferedOutputStream(OutputStream)	BufferedOutputStream(System.out)	BufferedOutputStream(FileOutputStream(String or File))	write(int) write(byte[])

Buffered Character Format

- Handle I/O of character data, automatically handling translation to and from the local character set.

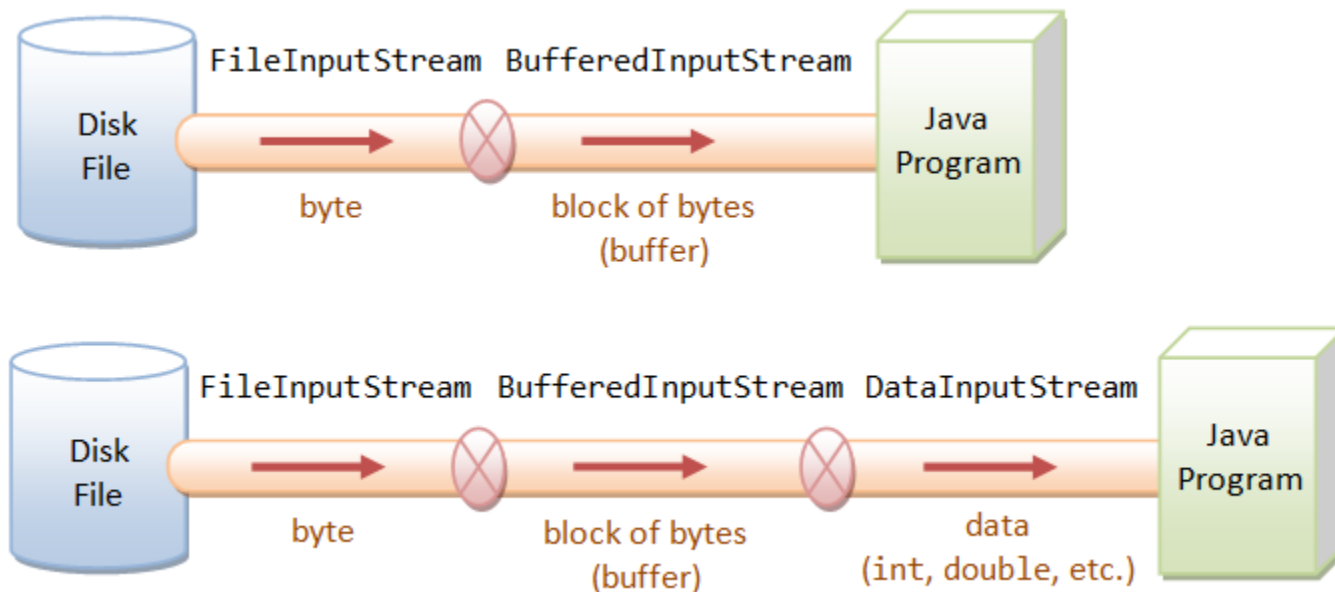
Base Class	Console	File	Method
BufferedReader(Reader)	BufferedReader(InputStreamReader(System.in))	BufferedReader(FileReader(String or File))	read() – return int readLine()
BufferedWriter(Writer)	BufferedWriter(OutputStreamWriter(System.out))	BufferedWriter(FileWriter(String or File))	write(String) newLine() flush()

Data Stream Format

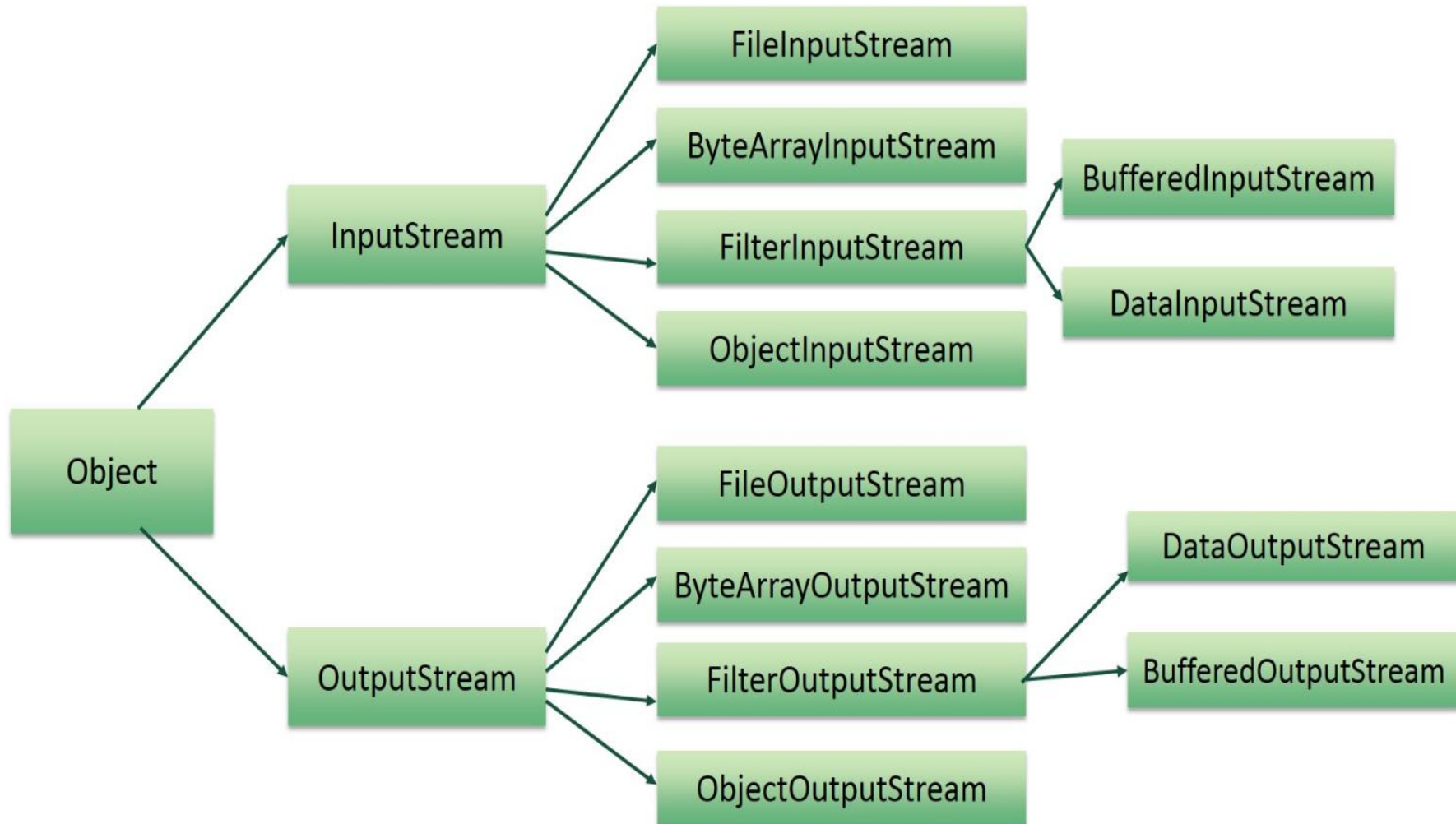
- Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values.

Base Class	Console	File	Method
DataInput(int erface) InputStream	DataInputStream (System.in) or DataInputStream (BufferedInputStream(System.in))	DataInputStream (FileInputStream) or DataInputStream (BufferedInputStream(FileInputStream))	int i = dis.read(); boolean b = dis.readBoolean(); String s1 = dis.readLine(); String s2 = dis.readUTF();
DataOutput(I nterface) OutputStream	DataOutputStream(Sys tem.out) or DataOutputStream(Buf feredOutputStream(Sy stem.out))	DataOutputStream (FileOutputStream) or DataOutputStream (BufferedOutputStrea m(FileOutputStream))	writeInt(int); writeUTF(String); writeByte(int); writeBoolean(boolean); write(int);

Pictorial representation of different format.



Stream Hierarchy



PrintWriter

- Is one of the character-based classes.
- Using a character-based class for console output makes internationalizing your program easier.
- `PrintWriter` contains higher level output methods to write data.
- `Print()` and `println()` prints any kind of data,
 - If an argument of `print/println` is not primitive type, the `PrintWriter` methods call the object's `toString()` method and then display the result.
- Like `PrintStream`, `PrintWriter`'s methods don't throw `IOException`.

PrintWriter

- Constructors:

- **PrintWriter(File file)**

- This creates a new PrintWriter, without automatic line flushing, with the specified file.

- **PrintWriter(OutputStream out)**

- This creates a new PrintWriter, without automatic line flushing, from an existing OutputStream.

- **PrintWriter(OutputStream out, boolean autoFlush) – most commonly used**

- This creates a new PrintWriter from an existing OutputStream

- **PrintWriter(String fileName)**

- This creates a new PrintWriter, without automatic line flushing, with the specified file name.

- **PrintWriter(Writer out)**

- This creates a new PrintWriter, without automatic line flushing.

- **PrintWriter(Writer out, boolean autoFlush)**

- This creates a new PrintWriter.

PrintWriter - methods

Modifier and Type	Method and Description
<u>PrintWriter</u>	<u>append</u> (char c)Appends the specified character to this writer.
void	<u>close</u> ()Closes the stream and releases any system resources associated with it.
void	<u>flush</u> ()Flushes the stream.
void	<u>print</u> (boolean b)Prints a boolean value.
void	<u>print</u> (char c)Prints a character.
void	<u>print</u> (char[] s)Prints an array of characters.
void	<u>print</u> (double d)Prints a double-precision floating-point number.
void	<u>print</u> (float f)Prints a floating-point number.
void	<u>print</u> (int i)Prints an integer.
void	<u>print</u> (long l)Prints a long integer.
void	<u>print</u> (<u>Object</u> obj)Prints an object.
void	<u>print</u> (<u>String</u> s)Prints a string.
<u>PrintWriter</u>	<u>printf</u> (<u>String</u> format, <u>Object</u> ... args)A convenience method to write a formatted string to this writer using the specified format string and arguments.
void	<u>println</u> ()Terminates the current line by writing the line separator string.
void	<u>println</u> (boolean x)Prints a boolean value and then terminates the line.
void	<u>println</u> (char x)Prints a character and then terminates the line.
void	<u>println</u> (char[] x)Prints an array of characters and then terminates the line.

PrintWriter - methods

Modifier and Type	Method and Description
void	<u>println</u> (double x) Prints a double-precision floating-point number and then terminates the line.
void	<u>println</u> (float x) Prints a floating-point number and then terminates the line.
void	<u>println</u> (int x) Prints an integer and then terminates the line.
void	<u>println</u> (long x) Prints a long integer and then terminates the line.
void	<u>println</u> (<u>Object</u> x) Prints an Object and then terminates the line.
void	<u>println</u> (<u>String</u> x) Prints a String and then terminates the line.
protected void	<u>setError</u> () Indicates that an error has occurred.
void	<u>write</u> (char[] buf) Writes an array of characters.
void	<u>write</u> (char[] buf, int off, int len) Writes A Portion of an array of characters.
void	<u>write</u> (int c) Writes a single character.
void	<u>write</u> (<u>String</u> s) Writes a string.
void	<u>write</u> (<u>String</u> s, int off, int len) Writes a portion of a string.

Example - Copy from a file

```
import java.io.*;
public class CopyFile {
    public static void main(String[] args) {
        File inFile = new File("input.txt");
        PrintWriter pw = null;
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(inFile));
            pw = new PrintWriter("output.txt");
            String data = br.readLine();
            while(data != null){
                pw.println(data);
                data = br.readLine();
            }
        } catch (IOException e1) {
            e1.printStackTrace();
        } finally{
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            pw.close();
        }
    }
}
```

Resource Management-autoclosing

- JDK 7 added a new feature to manage resources, such as file streams, by automating the closing process.
 - This feature is known as *automatic resource management*, or *ARM* for short, is based on an expanded version of the **try statement**.
- ***The principal advantage*** of automatic resource management is that
 - it prevents situations in which a file (or other resource) is inadvertently not released after it is no longer needed.
 - Forgetting to close a file can result in memory leaks, and could lead to other problems.

Resource Management-autoclosing

- General Form

```
try (resource-specification) {  
    // use the resource  
}
```

- The **try-with-resources** statement can be used only with those resources that implements
 - Either the `AutoCloseable` interface defined in `java.lang`
 - Or `Closeable`(subclass of `AutoClosable`) interface in `java.io`.
- Both interfaces are implemented by the stream classes.
- Thus, `try-with-resources` can be used when working any stream.

Example – Copy file(autoclosing)

```
import java.io.*;

public class CopyFile1 {
    public static void main(String[] args) {
        File inFile = new File("input.txt");
        File outFile = new File("output.txt");

        // br and pw will be automatically closed after try block exits
        try (BufferedReader br = new BufferedReader(new FileReader(inFile));
            PrintWriter pw = new PrintWriter(outFile)){
            String data = br.readLine();
            while(data != null){
                pw.println(data);
                data = br.readLine();
            }
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
```

File Class – in java.io package

- Java File class represents the files and directory pathnames in an abstract manner.
- This class is used for creation of files and directories, file searching, file deletion, etc.
 - But not for reading/writing to file.

Constructor and Description

File(**File** parent, **String** child) Creates a new File instance from a parent abstract pathname and a child pathname string.

File(**String** pathname) Creates a new File instance by converting the given pathname string into an abstract pathname.

File(**String** parent, **String** child) Creates a new File instance from a parent pathname string and a child pathname string.

File(**URI** uri) Creates a new File instance by converting the given file: URI into an abstract pathname.

File Class – in java.io package

Modifier and Type	Method and Description
boolean	<u>createNewFile()</u> Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	<u>delete()</u> Deletes the file or directory denoted by this abstract pathname.
boolean	<u>exists()</u> Tests whether the file or directory denoted by this abstract pathname exists.
<u>String</u>	<u>getAbsolutePath()</u> Returns the absolute pathname string of this abstract pathname.
<u>String</u>	<u>getName()</u> Returns the name of the file or directory denoted by this abstract pathname.
<u>String</u>	<u>getParent()</u> Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
<u>File</u>	<u>getParentFile()</u> Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
<u>String</u>	<u>getPath()</u> Converts this abstract pathname into a pathname string.

File Class – in java.io package

Modifier and Type	Method and Description
boolean	<u>isDirectory()</u> Tests whether the file denoted by this abstract pathname is a directory.
boolean	<u>isFile()</u> Tests whether the file denoted by this abstract pathname is a normal file.
<u>String[]</u>	<u>list()</u> Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
<u>File[]</u>	<u>listFiles()</u> Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
boolean	<u>mkdir()</u> Creates the directory named by this abstract pathname.
boolean	<u>mkdirs()</u> Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories.
boolean	<u>renameTo(<u>File</u> dest)</u> Renames the file denoted by this abstract pathname.
<u>Path</u>	<u>toPath()</u> Returns a <u>java.nio.file.Path</u> object constructed from the this abstract path.
<u>String</u>	<u>toString()</u> Returns the pathname string of this abstract pathname.

File I/O (Featuring NIO.2)

- The java.nio.file package
 - provide comprehensive support for file I/O and for accessing the default file system.
- Checking a File or Directory

Files Class -in java.nio.file package

Modifier and Type	Method and Description
static long	<u>copy</u> (<u>InputStream</u> in, <u>Path</u> target, <u>CopyOption</u> ... options)Copies all bytes from an input stream to a file.
static long	<u>copy</u> (<u>Path</u> source, <u>OutputStream</u> out)Copies all bytes from a file to an output stream.
static <u>Path</u>	<u>copy</u> (<u>Path</u> source, <u>Path</u> target, <u>CopyOption</u> ... options)Copy a file to a target file.
static <u>Path</u>	<u>createDirectories</u> (<u>Path</u> dir, <u>FileAttribute</u> <?>... attrs)Creates a directory by creating all nonexistent parent directories first.
static <u>Path</u>	<u>createDirectory</u> (<u>Path</u> dir, <u>FileAttribute</u> <?>... attrs)Creates a new directory.
static <u>Path</u>	<u>createFile</u> (<u>Path</u> path, <u>FileAttribute</u> <?>... attrs)Creates a new and empty file, failing if the file already exists.
static void	<u>delete</u> (<u>Path</u> path)Deletes a file.
static boolean	<u>deleteIfExists</u> (<u>Path</u> path)Deletes a file if it exists.
static boolean	<u>exists</u> (<u>Path</u> path, <u>LinkOption</u> ... options)Tests whether a file exists.
static <u>Object</u>	<u>getAttribute</u> (<u>Path</u> path, <u>String</u> attribute, <u>LinkOption</u> ... options)Reads the value of a file attribute.
static <V extends <u>FileAttributeVi <u>ew</u></u> > V	<u>getFileAttributeView</u> (<u>Path</u> path, <u>Class</u> <V> type, <u>LinkOption</u> ... options)Returns a file attribute view of a given type.

Files Class -in java.nio.file package

Modifier and Type	Method and Description
static boolean	<u>isDirectory</u> (<u>Path</u> path, <u>LinkOption</u> ... options)Tests whether a file is a directory.
static boolean	<u>isSameFile</u> (<u>Path</u> path, <u>Path</u> path2)Tests if two paths locate the same file.
static <u>Path</u>	<u>move</u> (<u>Path</u> source, <u>Path</u> target, <u>CopyOption</u> ... options)Move or rename a file to a target file.
static <u>BufferedReader</u>	<u>newBufferedReader</u> (<u>Path</u> path, <u>Charset</u> cs)Opens a file for reading, returning a <u>BufferedReader</u> that may be used to read text from the file in an efficient manner.
static <u>BufferedWriter</u>	<u>newBufferedWriter</u> (<u>Path</u> path, <u>Charset</u> cs, <u>OpenOption</u> ... options)Opens or creates a file for writing, returning a <u>BufferedWriter</u> that may be used to write text to the file in an efficient manner.
static <u>InputStream</u>	<u>newInputStream</u> (<u>Path</u> path, <u>OpenOption</u> ... options)Opens a file, returning an input stream to read from the file.
static <u>OutputStream</u>	<u>newOutputStream</u> (<u>Path</u> path, <u>OpenOption</u> ... options)Opens or creates a file, returning an output stream that may be used to write bytes to the file.
static boolean	<u>notExists</u> (<u>Path</u> path, <u>LinkOption</u> ... options)Tests whether the file located by this path does not exist.
static byte[]	<u>readAllBytes</u> (<u>Path</u> path)Reads all the bytes from a file.
static <u>List</u> < <u>String</u> >	<u>readAllLines</u> (<u>Path</u> path, <u>Charset</u> cs)Read all lines from a file.

Reference

- Java: Complete Reference - Chapter 13, 20, 21