UNIT 3 –I/O PROGRAMMING

Java IO streams

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

Java IO streams

Stream

- A stream is a sequence of data.
- In Java, a stream is composed of bytes.
- It's called a stream because it is like a stream of water that continues to flow.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
    1. System.out: standard output stream
    2. System.in: standard input stream
    3. System.err: standard error stream

Java IO streams

int i=System.in.read(); //returns ASCII code of 1st character

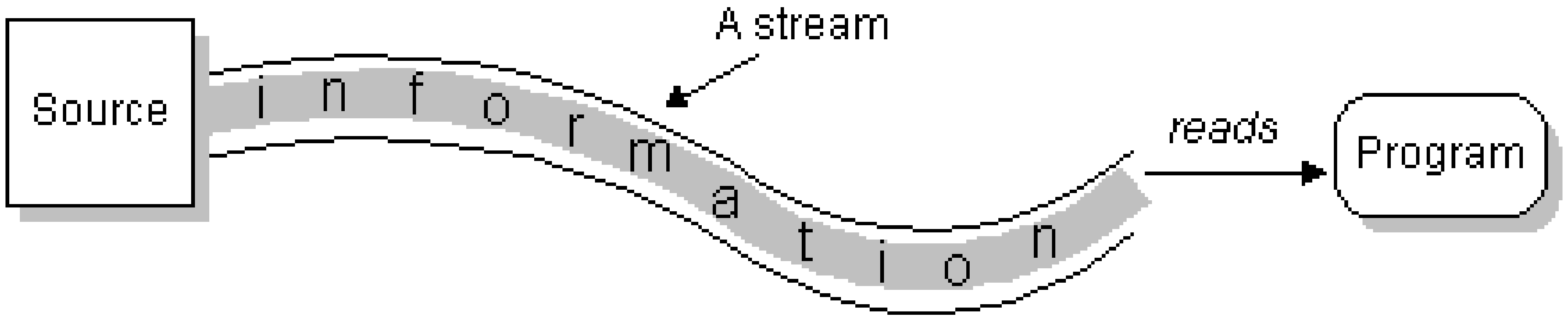
System.out.println((char)i); //will print the character

System.out.println("simple message"); //print output message to console

System.err.println("error message"); //print an error message to the console

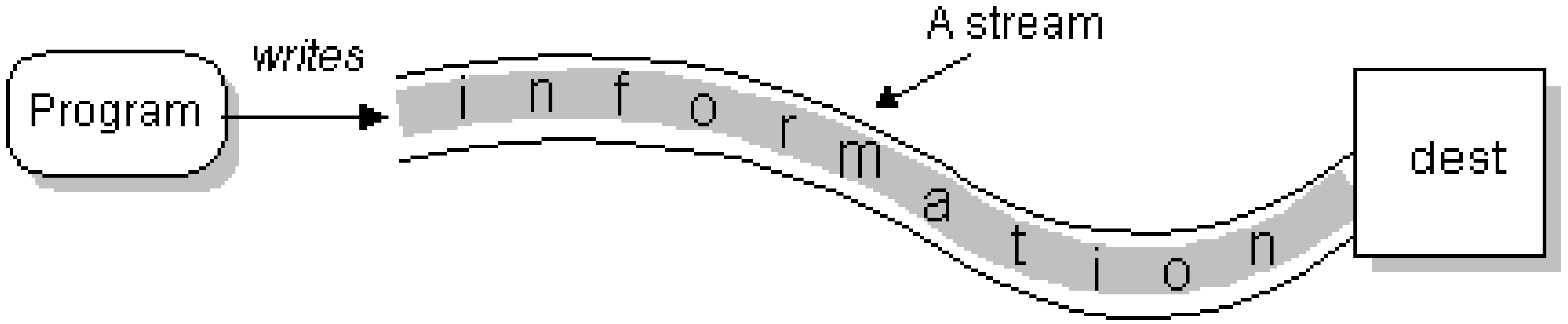
Java IO streams

- To bring in information, a program opens a stream on an information source (a file, memory, a socket) and reads the information sequentially, as shown here:



Java IO streams

- Similarly, a program can send information to an external destination by opening a stream to a destination and writing the information out sequentially, like this:



Java IO streams

- No matter where the data is coming from or going to and no matter what its type, the algorithms for sequentially reading and writing data are basically the same:

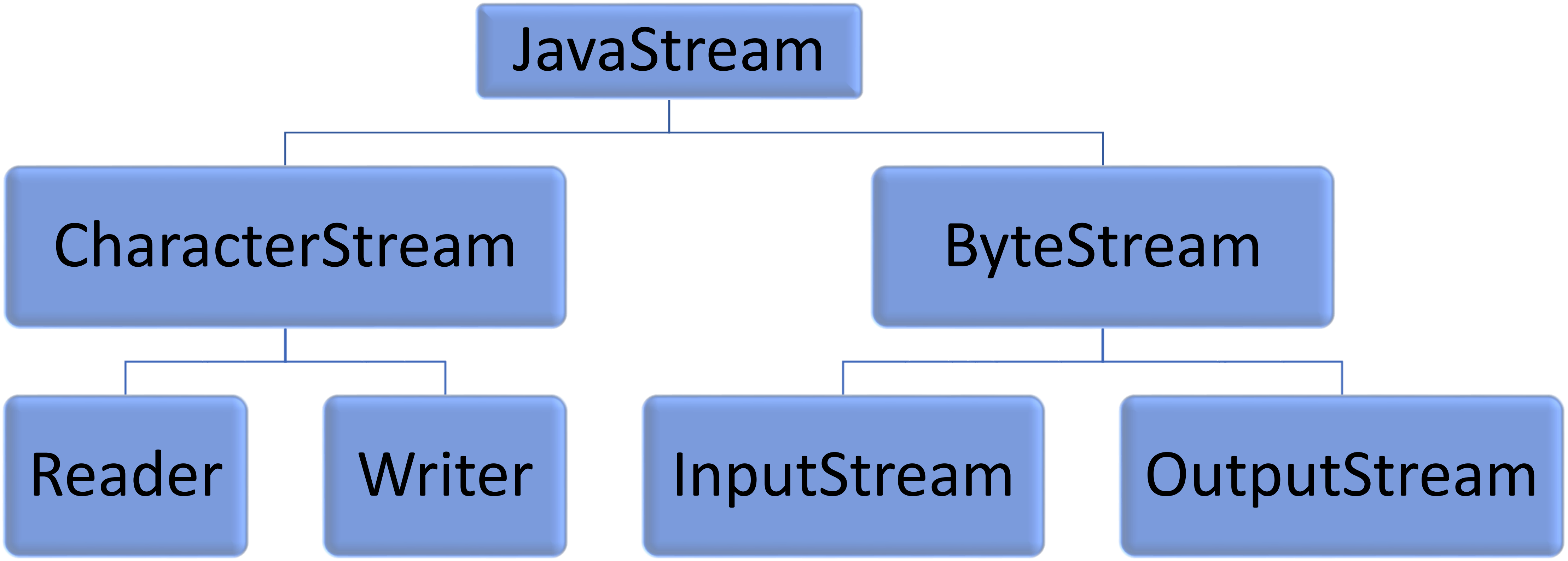
- Reading

```
open a stream
while more information
    read information
close the stream
```

- Writing

```
open a stream
while more information
    write information
close the stream
```

Java IO streams



Character and Binary streams

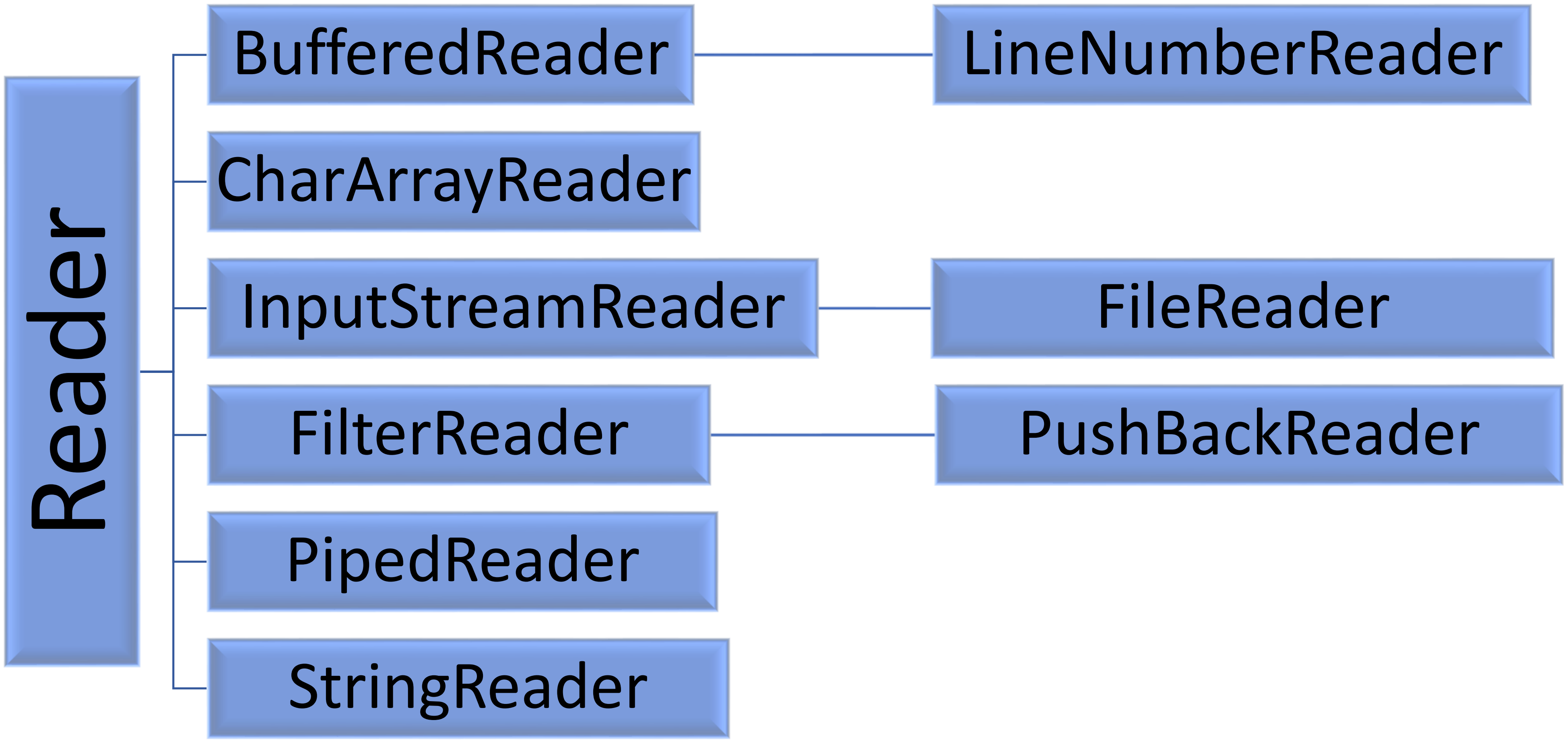
Character Stream Classes

- Character stream can be used to read and write 16-bit Unicode characters.
- Reader and Writer are the abstract superclasses for character streams in java.io.
- Most programs should use readers and writers to read and write textual information. The reason is that they can handle any character in the Unicode character set, whereas the byte streams are limited to ISO-Latin-1 8-bit bytes.

CharacterStream: Reader

- Reader provides the API and partial implementation for readers--streams that read 16-bit characters
- Reader stream classes are designed to read character from the files.
- The reader class contains methods that are identical to those available in the InputStream class, except Reader is designed to handle characters. Therefore, reader classes can perform all the functions implements by the input stream classes.
- The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.

CharacterStream: Reader Hierarchy



CharacterStream: Reader

- Useful methods of InputStream

Modifier and Type	**Method**	**Description**
abstract void	close()	It closes the stream and releases any system resources associated with it.
void	mark(int readAheadLimit)	It marks the present position in the stream.
int	read()	It reads a single character.
abstract int	read(char[] cbuf, int off, int len)	It reads characters into a portion of an array.
boolean	ready()	It tells whether this stream is ready to be read.
void	reset()	It resets the stream.
long	skip(long n)	It skips characters.

CharacterStream: Reader

```
import java.io.*;
public class ReaderExample {
   public static void main(String[] args) {
      try {
         Reader reader = new FileReader("file.txt");
         int data = reader.read();
         while (data != -1) {
            System.out.print((char) data);
            data = reader.read();
         }
         reader.close();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
}
```

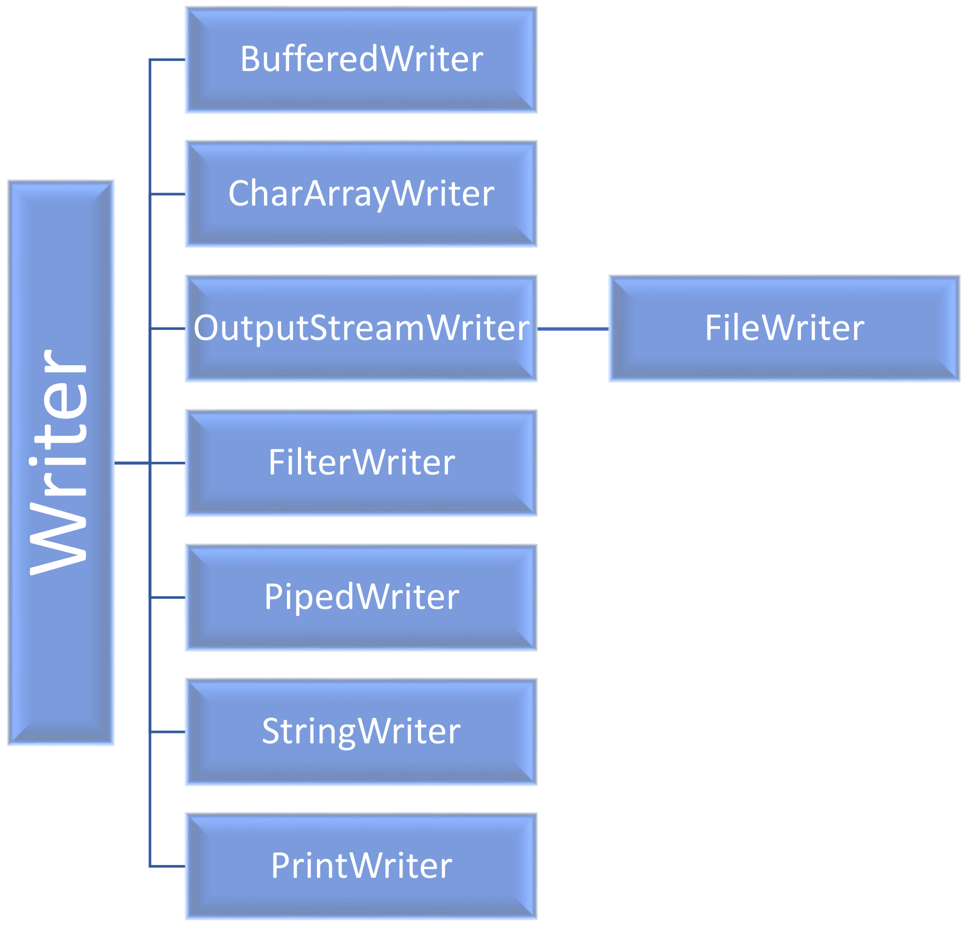
file.txt
I love my country

Output:
I love my country

CharacterStream: Writer

- Writer provides the API and partial implementation for writers--streams that write 16-bit characters.
- It is an abstract class for writing to character streams.
- The methods that a subclass must implement are write(char[], int, int), flush(), and close().
- Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

CharacterStream: Writer Hierarchy



CharacterStream: Writer

- Useful methods of InputStream

Modifier and Type	Method	Description
Writer	append(char c)	It appends the specified character to this writer.
Writer	append(CharSequence csq)	It appends the specified character sequence to this writer
Writer	append(CharSequence csq, int start, int end)	It appends a subsequence of the specified character sequence to this writer.
abstract void	close()	It closes the stream, flushing it first.
abstract void	flush()	It flushes the stream.
void	write(int c)	It writes a single character.
void	write(String str, int off, int len)	It writes a portion of a string.

CharacterStream: Writer

```
import java.io.*;
public class WriterExample {
    public static void main(String[] args) {
        try {
            Writer w = new FileWriter("output.txt");
            String content = "I love my country";
            w.write(content);
            w.close();
            System.out.println("Done");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:
Done

Output.txt
I love my country

Byte Stream Classes

- Byte Stream Classes have been designed to provide functional features for creating and manipulating streams and files for reaing and writing bytes.
- Since the streams are unidirectional, they can transmit bytes in only one direction and, therefore, java Provides two kinds of byte stream classes:
  - Input Stream Classes
  - Output Stream Classes

ByteStream: InputStream

- Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.
- Input stream classes are used to read 8-bit bytes.

InputStream class

- InputStream class is an abstract class. Therefore, we can not create instances of this class rather we must use the subclasses that inherits from this class.
- It is the superclass of all classes representing an input stream of bytes.

ByteStream: InputStream

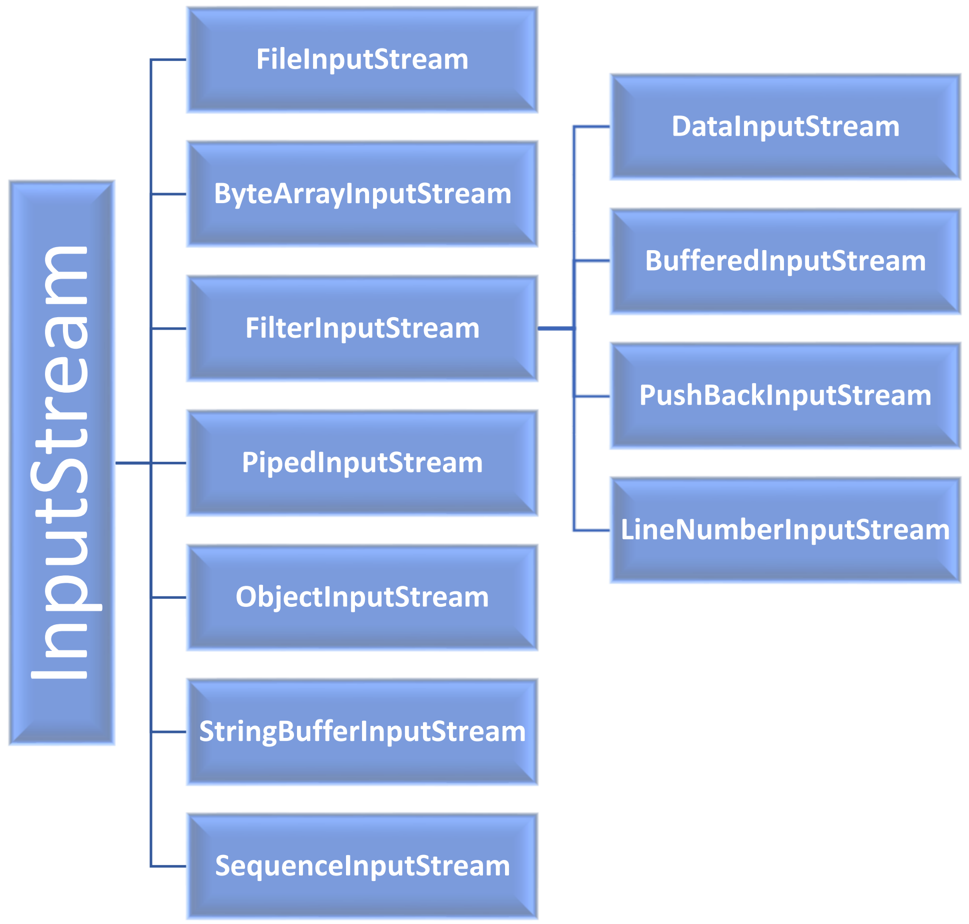
- InputStream class defines methods for performing input functions such as:
  - Reading Bytes
  - Closing streams
  - Marking positions in streams
  - Skipping ahead in a stream
  - Finding the number of bytes in a stream

ByteStream: InputStream

- Useful methods of InputStream

Method	**Description**
public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
public int skip(n)throws IOException	Skips over n bytes from the input stream
public int reset()throws IOException	Goes back to the beginning of the stream
public void close()throws IOException	is used to close the current input stream.

ByteStream: InputStream Hierarchy



Note: DataInputstream extends FilterInputStream and implements interface DataInput. Therefore, the DataInputStream implements the methods described in DataInput in addition to using the methods of InputStream

ByteStream: OutputStream

- Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

OutputStream class

- OutputStream class is an abstract class and therefore we cannot instantiate it.
- It is the superclass of all classes representing an output stream of bytes.
- An output stream accepts output bytes and sends them to some sink.

ByteStream: OutputStream

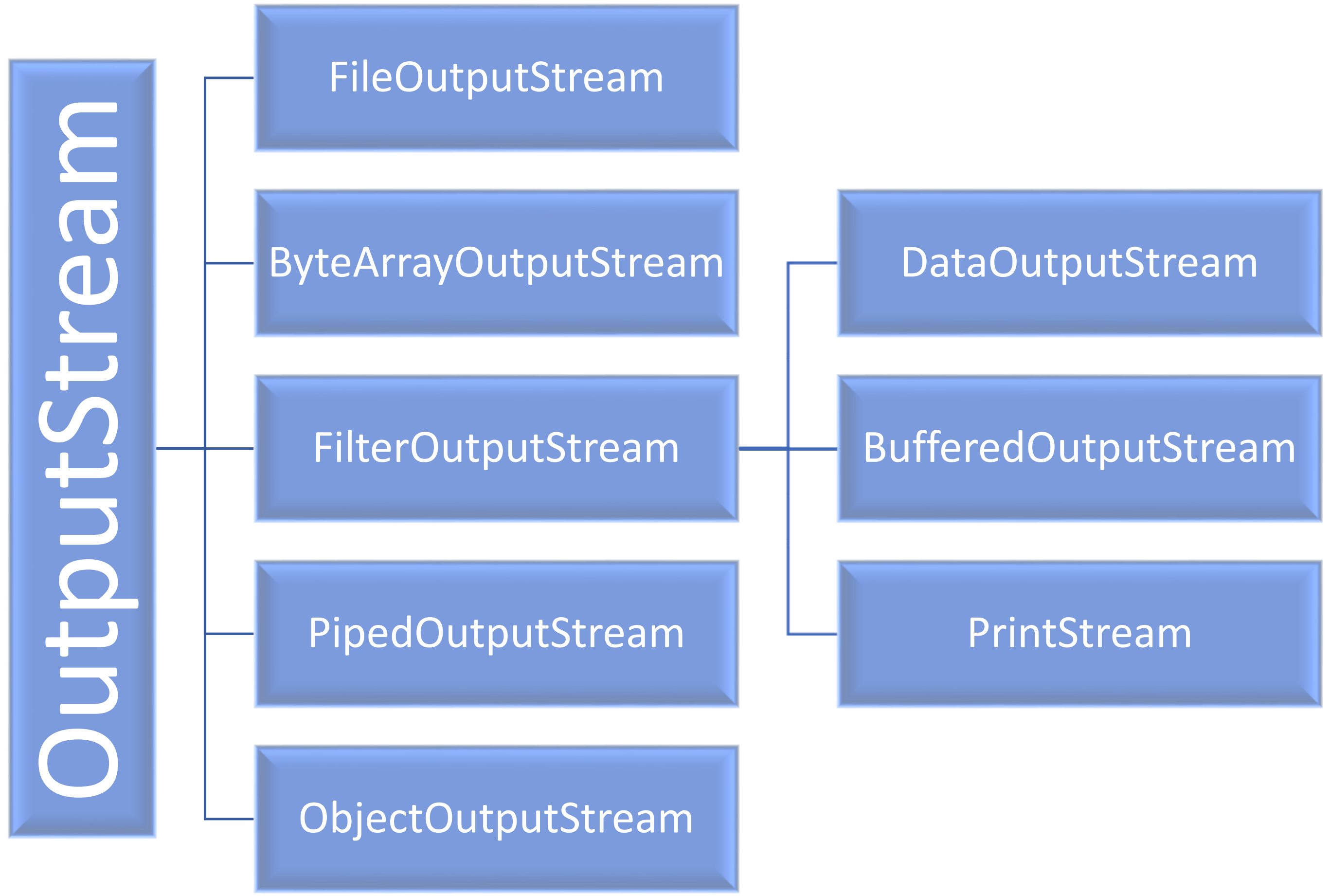
- OutputStream class defines methods for performing input functions such as:
  - Writing Bytes
  - Closing streams
  - Flushing streams

ByteStream: OutputStream

- Useful methods of OutputStream

Method	**Description**
public void write(int)throws IOException	is used to write a byte to the current output stream.
public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
public void flush()throws IOException	flushes the current output stream.
public void close()throws IOException	is used to close the current output stream.

ByteStream: OutputStream Hierarchy



Note: DataOutputstream implements interface DataOutput and, therefore, implements the methods described in DataOutput interface.

Reading data from and writing data to files

Reading data from files

You can read files using these classes:

- **FileReader class (from CharacterStream)** is used to read data from the file. It returns data in byte format like FileInputStream class. It is character-oriented class which is used for file handling in java.

- **FileInputStream class (from ByteStream)** obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

Reading data from files: FileReader

Constructor	**Description**
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Method	**Description**
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Reading data from files: FileReader

```
import java.io.FileReader;
public class FileReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\file.txt");
        int i;
        while((i=fr.read())!=-1)
        System.out.print((char)i);
        fr.close();
   }
}
```

file.txt
I love my country

Output:
I love my country

Reading data from files: FileInputStream

Method	**Description**
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to close the stream.

Reading data from files: FileInputStream

```
// example to read single character
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\file.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

file.txt
I love my country

Output:
I

Reading data from files: FileInputStream

```
// example to read all characters
import java.io.FileInputStream;
public class DataStreamExample {
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("D:\\file.txt");  //Exception-1
          int i=0;
          while((i=fin.read())!=-1){    //Exception-2
           System.out.print((char)i); //Exception-2
          }
          fin.close();          //Exception-3
        }catch(Exception e){System.out.println(e);}
    }
}
```

file.txt
I love my country

Output:
I love my country

Writing data to files

You can write files using these classes:

- **FileWriter class (from CharacterStream)** is used to write character-oriented data to a file. It is character-oriented class which is used for file handling in java.

- **FileOutputStream class (from ByteStream)** is an output stream used for writing data to a file. If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

Writing data to files: FileWriter

Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

FileWriter

```
import java.io.FileWriter;
public class FileWriterExample {
    public static void main(String args[]){
         try{
           FileWriter fw=new FileWriter("D:\\file.txt");
           fw.write("Welcome to Java");
           fw.close();
          }catch(Exception e){
          System.out.println(e);}
          System.out.println("Success...");
     }
}
```

Output:
Success...

file.txt
Welcome to Java

Writing data to files: FileOutputStream

Method	**Description**
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Writing data to files: FileOutputStream

```
//example to write byte
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
         try{
           FileOutputStream fout=new FileOutputStream("D:\\file.txt");
           fout.write(65);
           fout.close();
           System.out.println("success...");
          }catch(Exception e){System.out.println(e);}
     }
}
```

Output:
Success...

file.txt
A

Writing data to files: FileOutputStream

```
//example to write string
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
         try{
           FileOutputStream fout=new FileOutputStream("D:\\file.txt");
           String s="Welcome to Java";
           byte b[]=s.getBytes();//converting string into byte array
           fout.write(b);
           fout.close();
           System.out.println("success...");
          }catch(Exception e){System.out.println(e);}
     }
}
```

Output:
Success...

file.txt
Welcome to Java

Reading data from files: BufferedReader

- Data can be read line by line using BufferedReader class
  - to read the text from a character-based input stream
- It uses buffer-makes the performance fast
- It inherits Reader class
- requires reader object as argument

Java BufferedReader class constructors

Constructor	Description
BufferedReader(Reader rd)	It is used to create a buffered character input stream that uses the default size for an input buffer.
BufferedReader(Reader rd, int size)	It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

Java BufferedReader Example

```
import java.io.*;
public class BufferedReaderExample {
    public static void main(String args[])throws Exception{
          FileReader fr=new FileReader("D:\\testout.txt");
          BufferedReader br=new BufferedReader(fr);

          int i;
          while((i=br.read())!=-1){
          System.out.print((char)i);
          }
          br.close();    //sequence is reverse
          fr.close();
    }
}
```

Writing data into files: BufferedWriter

- Used to provide buffering for Writer instances
- It makes the performance fast
- It inherits Writer class

Java BufferedWriter class constructors

Constructor	Description
BufferedWriter (Writer wrt)	It is used to create a buffered character output stream that uses the default size for an output buffer.
BufferedWriter (Writer wrt, int size)	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Java BufferedWriter class methods

Method	Description
void newLine()	It is used to add a new line by writing a line separator.
void write(int c)	It is used to write a single character.
void write(char[] cbuf, int off, int len)	It is used to write a portion of an array of characters.
void write(String s, int off, int len)	It is used to write a portion of a string.
void flush()	It is used to flushes the input stream.
void close()	It is used to closes the input stream

Java BufferedWriter Example

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {
    FileWriter writer = new FileWriter("D:\\testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to JavaTpoint.");
    buffer.close();
    System.out.println("Success");
    }
}
```

Explore

- Reading data from console by InputStreamReader and BufferedReader

Serializable interface

- Serialization:
- To convert the state of an object into a byte stream –stores metadata in binary form
- done using ObjectOutputStream
- a mechanism of converting the state of an object into a byte stream
- Deserialization:
- The reverse process of serialization
- Byte stream is used to recreate the actual Java object in memory

Serializable interface

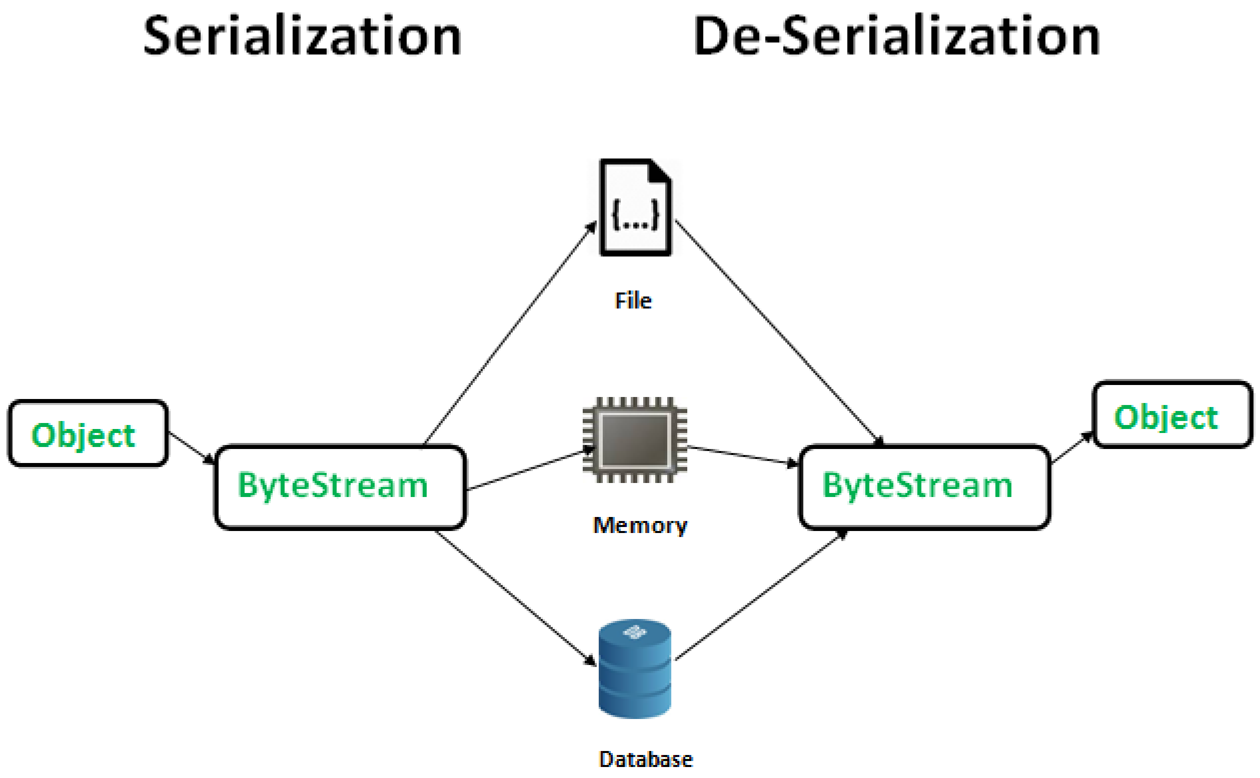


Image source: https://www.geeksforgeeks.org/serializable-interface-in-java/

Reading data from and Writing data to files on random Positions

RandomAccessFile

- This class is used for reading and writing to random access file.
- A random access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Reading data from and Writing data to files

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class RandomAccessFileExample
{
   static final String FILEPATH
="myFile.TXT";
   public static void main(String[] args) {
      try {
         System.out.println(new
String(readFromFile(FILEPATH, 0, 18)));
         writeToFile(FILEPATH, "I love my
country and my people", 31);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
```

```
private static byte[] readFromFile(String
filePath, int position, int size)
         throws IOException {
      RandomAccessFile file = new
RandomAccessFile(filePath, "r");
      file.seek(position);
      byte[] bytes = new byte[size];
      file.read(bytes);
      file.close();
      return bytes;
   }
```

```
 private static void writeToFile(String filePath,
String data, int position)
         throws IOException {
      RandomAccessFile file = new
RandomAccessFile(filePath, "rw");
      file.seek(position);
      file.write(data.getBytes());
      file.close();
   }
}
```

Output:
The myFile.TXT contains text "This class is used for reading and writing to random access file."
after running the program it will contains
This class is used for reading I love my country and my people.