UNIT I - JAVA FUNDAMENTALS

- Java Data types
- Class – Object
- I / O Streams
- File Handling concepts
- Threads
- Applets
- Swing Framework
- Reflection

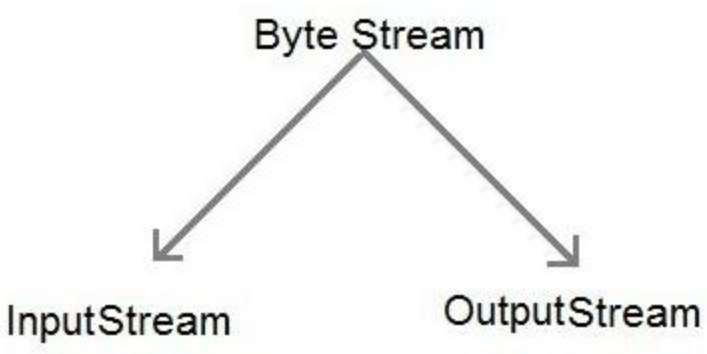
Presented by,
B.Vijayalakshmi
Computer Centre
MIT Campus
Anna University

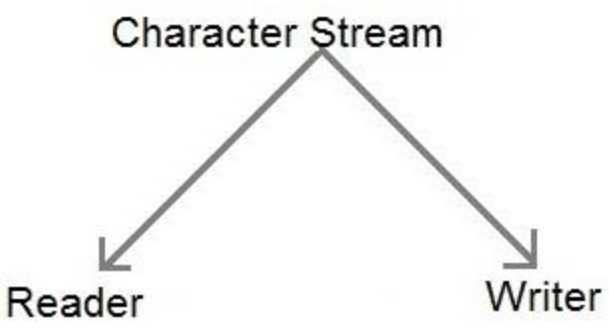
Streams

- A stream is a **sequence of data of undetermined length.**
- Java uses the concept of stream to **make I/O operation fast.**
- The **java.io package** contains all the classes required for input and output operations.
- In java, 3 streams are created for us automatically. All these streams are attached with console.
    - **System.out:** standard output stream
    - **System.in:** standard input stream
    - **System.err:** standard error stream

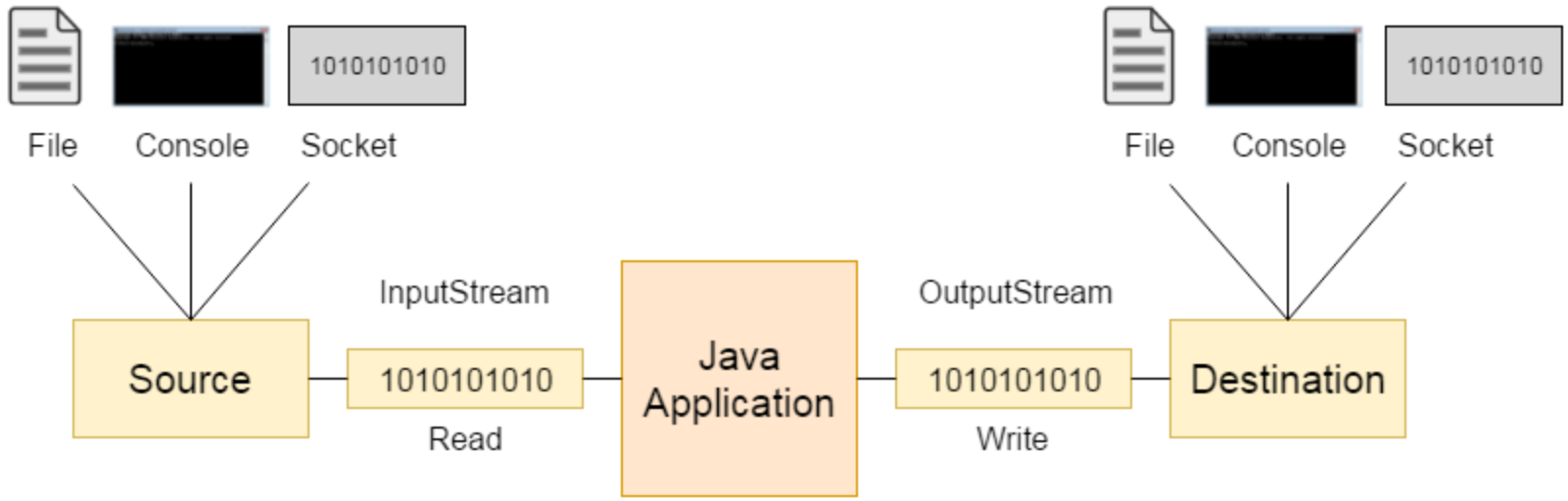
IO Stream

- Java defines two types of streams. They are,
  - **Byte Stream :** It provides a convenient means for **handling input and output of byte**.
  - **Character Stream :** It provides a convenient means for **handling input and output of characters.**





OutputStream vs InputStream

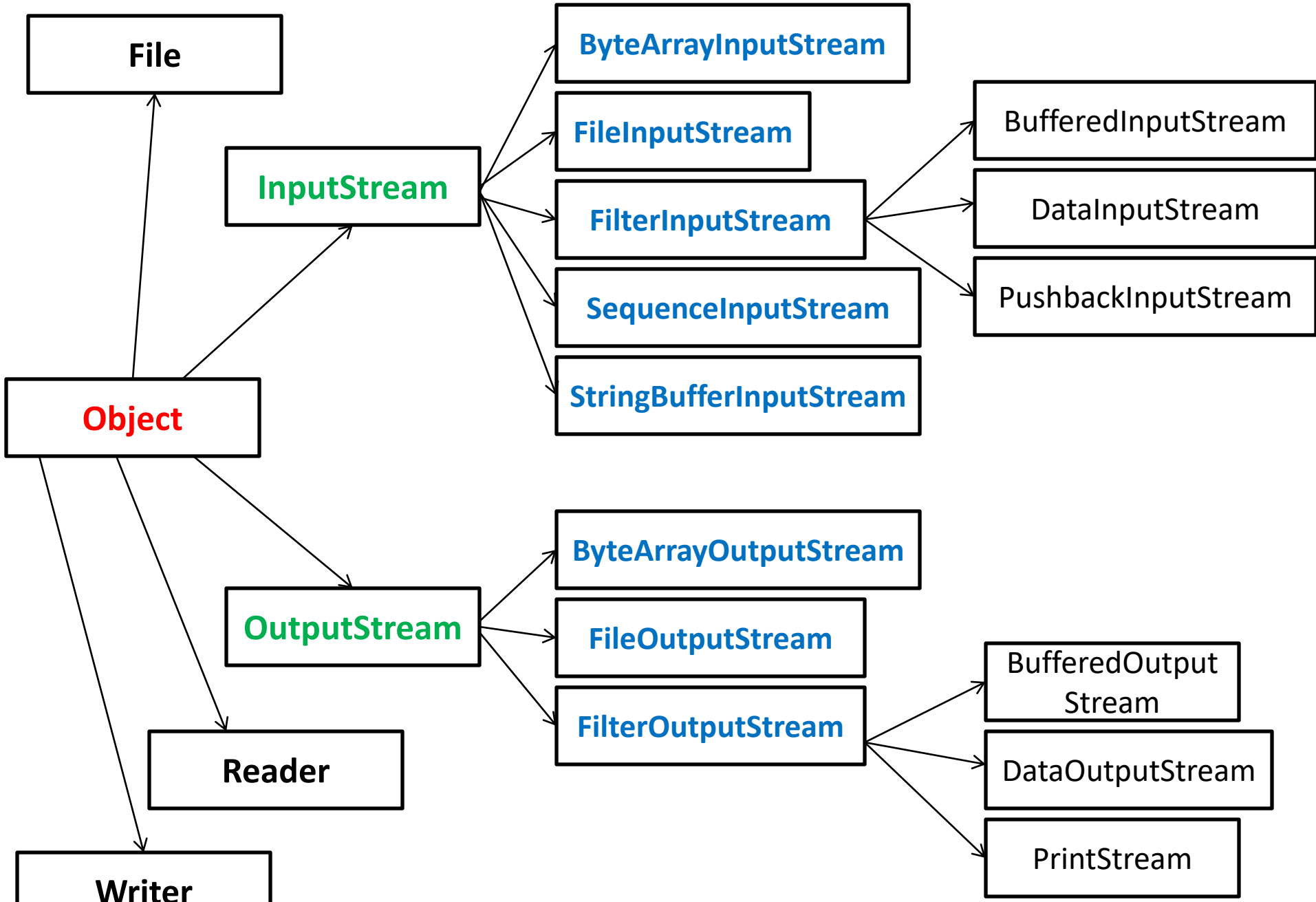


- InputStream

  Java application uses an **input stream to read data** from a source, it may be a file, an array, peripheral device or socket.

- OutputStream

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


File
InputStream
ByteArrayInputStream
FileInputStream
FilterInputStream
SequenceInputStream
StringBufferInputStream
BufferedInputStream
DataInputStream
PushbackInputStream
Object
OutputStream
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
BufferedOutput Stream
DataOutputStream
PrintStream
Reader
Writer

FileStreams

- **FileInputStream Class**
  - This class 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.
  - We can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.
- **Constructors**

1. InputStream f = new FileInputStream("C:/java/hello");
2. File f = new File("C:/java/hello");

   InputStream f = new FileInputStream(f);

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 starting at offset **off**
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 closes the stream.

FileStreams

- **FileOutputStream Class**
  - This is an output stream used for **writing data to a file**.
  - It can be used to **write primitive values and byte oriented data** into a file
  - It can be used for character-oriented data .But, it is preferred to use FileWriter than FileOutputStream.
- **Constructors**

1.OutputStream f = new FileOutputStream("C:/java/hello“)

2.File f = new File("C:/java/hello");

OutputStream f = new FileOutputStream(f);

Method	Description
protected void finalize()	It is sued 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.

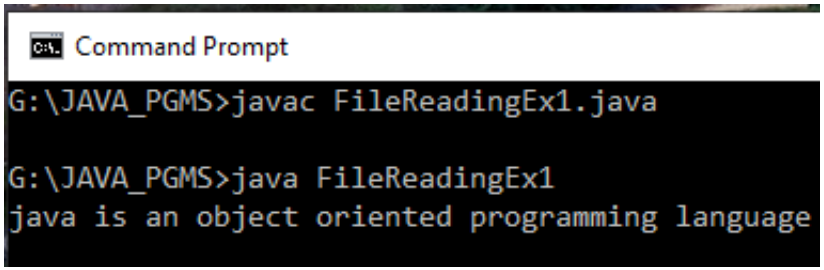
Reading from a file Example

//Read the content from already existing file and show in the screen

```
import java.io.*;
class FileReadingEx1
{
 public static void main(String a[])
 {
  try
  {
        int i=0;
        FileInputStream fin=new FileInputStream("test.txt");
        while((i=fin.read())!=-1)
        {
                System.out.print((char)i);
        }
        fin.close();
   }catch(Exception e)
    {
        System.out.println("Exception:"+e);
    }
 }
}
```

test.txt

java is an object oriented programming language



Writing to a file Example

//Get input from the user through keyboard an write into a file

```
import java.util.Scanner;
import java.io.*;
class FileWritingEx1
{
 public static void main(String a[])
 {
   try
   {
      byte b[]=new byte[100];
      char ch;
      int i=0;
      InputStreamReader in=new InputStreamReader(System.in);
      BufferedReader bs=new BufferedReader(in);
      FileOutputStream fop=new FileOutputStream(“abc.txt");
      System.out.println("Enter text you want to write in a file and end with # symbol:");
      while((ch=(char)bs.read())!='#')
      {
            b[i]=(byte)ch;
            i++;
      }
      fop.write(b);
      System.out.println("Successfully written in file...");
      fop.close();
 }
```

```
catch(Exception e)
                    {
                                System.out.println("Exception:"+e);
                    }
          }
}
```

Command Prompt

```
G:\JAVA_PGMS>javac FileWritingEx1.java

G:\JAVA_PGMS>java FileWritingEx1
Enter text you want to write in a file and end with # symbol:
Welcome to Web Tech Class
Best Wishes...#
Successfully written in file...
```

abc.txt - Notepad

File Edit Format View Help

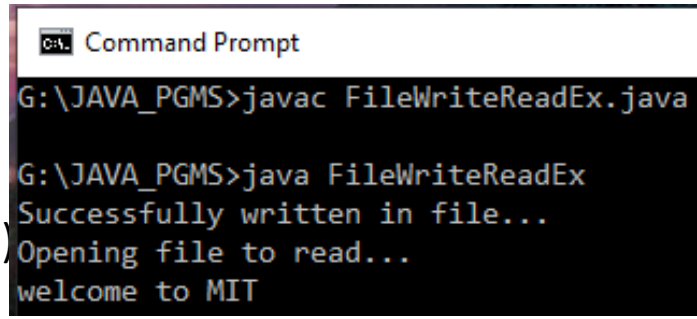
Welcome to Web Tech Class
Best Wishes...

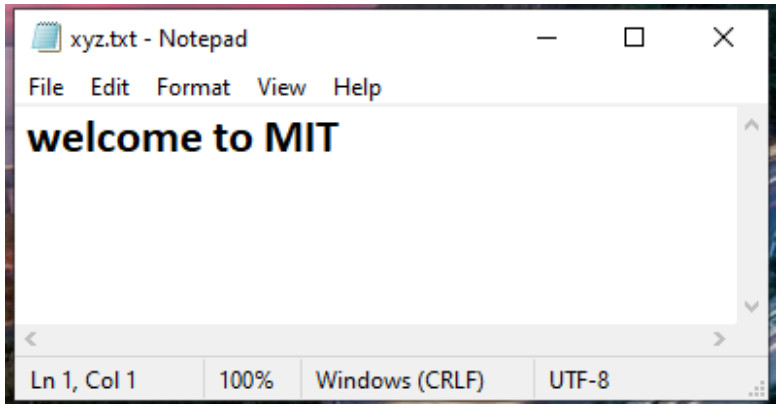
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Reading and Writing in a file example

//Convert a String data to byte data and write into a file and from that file read the data show in the screen

```
import java.util.Scanner;
import java.io.*;
class FileWriteReadEx
{
 public static void main(String a[])
 {   try
     {
            String str="welcome to MIT";
            FileOutputStream fop=new FileOutputStream("xyz.txt");
            byte b[]=str.getBytes();
            fop.write(b);
            System.out.println("Successfully written in file...");
            fop.close();
            FileInputStream fin=new FileInputStream("xyz.txt");
            System.out.println("Opening file to read...");
            int size=fin.available();
            for(int i=0;i<size;i++)
                System.out.print((char)fin.read());
            fin.close();
      }catch(Exception e)
       {
            System.out.println("Exception:"+e);
       }
 }
}
```





ByteArrayStream

- This Stream contains,
  - ➢ ByteArrayInputStream
  - ➢ ByteArrayOutputStream
- ByteArrayInputStream Class

  It can be used to read byte array as input stream.
- **Constructor**

1. ByteArrayInputStream(byte [] a)

This constructor accepts a byte array as a parameter.

2.ByteArrayInputStream(byte [] a, int off, int len)

This constructor takes an array of bytes, and two integer values, where **off** is the first byte to be read and **len** is the number of bytes to be read.

Methods	Description
int available()	It is used to return the number of remaining bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] ary, int off, int len)	It is used to read up to len bytes of data from an array of bytes in the input stream.
boolean markSupported()	It is used to test the input stream for mark and reset method.
long skip(long x)	It is used to skip the x bytes of input from the input stream.
void mark(int readAheadLimit)	It is used to set the current marked position in the stream.
void reset()	It is used to reset the buffer of a byte array.
void close()	It is used for closing a ByteArrayInputStream.

ByteArrayStream

ByteArrayOutputStream

- It creates a buffer in memory and all the data sent to the stream is stored in the buffer.
- The buffer of ByteArrayOutputStream automatically grows according to data.
- In this stream**, the data is written into a byte array which can be written to multiple streams later.**
- **Constructor**

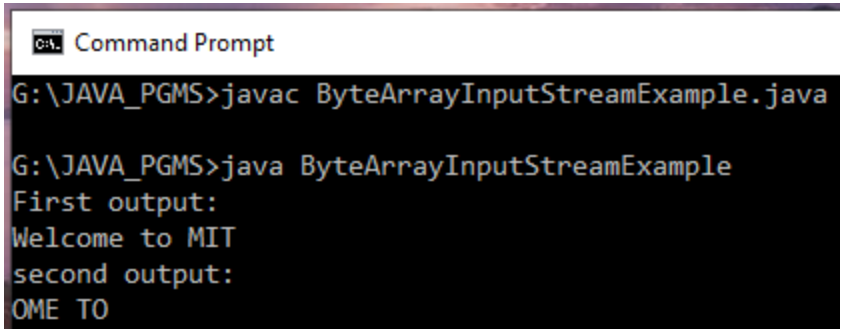
1. **ByteArrayOutputStream() -** Creates a new byte array output stream with the initial capacity of 32 bytes, though its size increases if necessary.
2. **ByteArrayOutputStream(int size)** - Creates a new byte array output stream, with a buffer capacity of the specified size, in bytes.

Method	Description
int size()	It is used to returns the current size of a buffer.
byte[] toByteArray()	It is used to create a newly allocated byte array.
String toString()	It is used for converting the content into a string decoding bytes using a platform default character set.
String toString(String charsetName)	It is used for converting the content into a string decoding bytes using a specified charsetName.
void write(int b)	It is used for writing the byte specified to the byte array output stream.
void write(byte[] b, int off, int len	It is used for writing **len** bytes from specified byte array starting from the offset **off** to the byte array output stream.
void writeTo(OutputStream out)	It is used for writing the complete content of a byte array output stream to the specified output stream.
void reset()	It is used to reset the count field of a byte array output stream to zero value.
void close()	It is used to close the ByteArrayOutputStream.

ByteArrayInputStreamExample

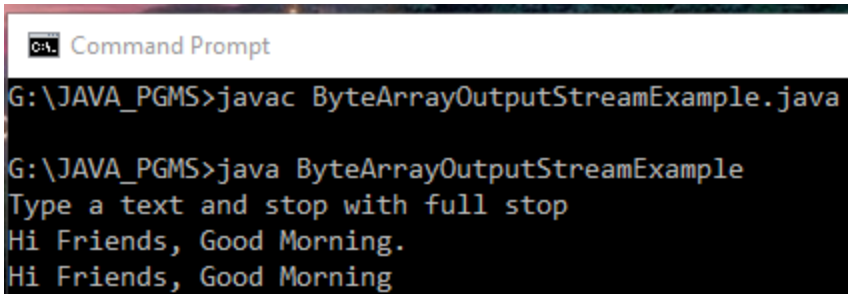
```
import java.io.*;

public class ByteArrayInputStreamExample
{

    public static void main(String[] args)
    {
        try
        {
            String str = "Welcome to MIT";

            //get bytes from string using getBytes method
            byte[] bytes = str.getBytes();

            //create ByteArrayInputStream object
            ByteArrayInputStream bip1 = new ByteArrayInputStream(bytes);
            ByteArrayInputStream bip2 = new ByteArrayInputStream(bytes,4,6);
```

```
    int ch, i;
    System.out.println("First output:");
    //read bytes from ByteArrayInputStream using read method
    while((ch = bip1.read()) != -1)
    {
            System.out.print((char)ch);
     }

    System.out.println("\nsecond output:");
     while((i=bip2.available())>0)
      {
            ch = bip2.read();
            System.out.print(Character.toUpperCase((char)ch));
       }
    }catch(Exception e)
            {
                System.out.print("Exception:"+e);
            }
  }
}
```



ByteArrayOutputStreamExample

```
import java.io.*;
 public class ByteArrayOutputStreamExample
{
    public static void main(String[] args)
   {
      try
     {
            ByteArrayOutputStream bop = new ByteArrayOutputStream();
            System.out.println("Type a text and stop with full stop");
            char ch;
            while((ch=(char)System.in.read())!='.')
            {
                        bop.write(ch);
            }
            byte b[]=bop.toByteArray();
            for(int l=0;l<b.length;l++)
                        System.out.print((char)b[l]);
        }catch(Exception e)
            {
                        System.out.print("Exception:"+e);
            }
     }
}
```



ByteArrayOutputStreamExample2

```
import java.io.*;
 public class ByteArrayOutputStreamExample2
{
    public static void main(String[] args)
    {
        try
        {
            FileOutputStream fout1=new FileOutputStream("f1.txt");
            FileOutputStream fout2=new FileOutputStream("f2.txt");
            String str="welcome to MIT";
            byte b[]=str.getBytes();
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            bout.write(b);
            bout.writeTo(fout1);
            bout.reset();
            bout.write(b,2,4);
            bout.writeTo(fout2);
            bout.close();
            System.out.println("successfully written to files...");
        }catch(Exception e)
            {
                System.out.print("Exception:"+e);
            }
    }
}
```

Command Prompt

```
G:\JAVA_PGMS>javac ByteArrayOutputStreamExample2.java

G:\JAVA_PGMS>java ByteArrayOutputStreamExample2
successfully written to files...
```

f1.txt - Notepad

File Edit Format View Help

welcome to MIT

100% Windows (CRLF) UTF-8

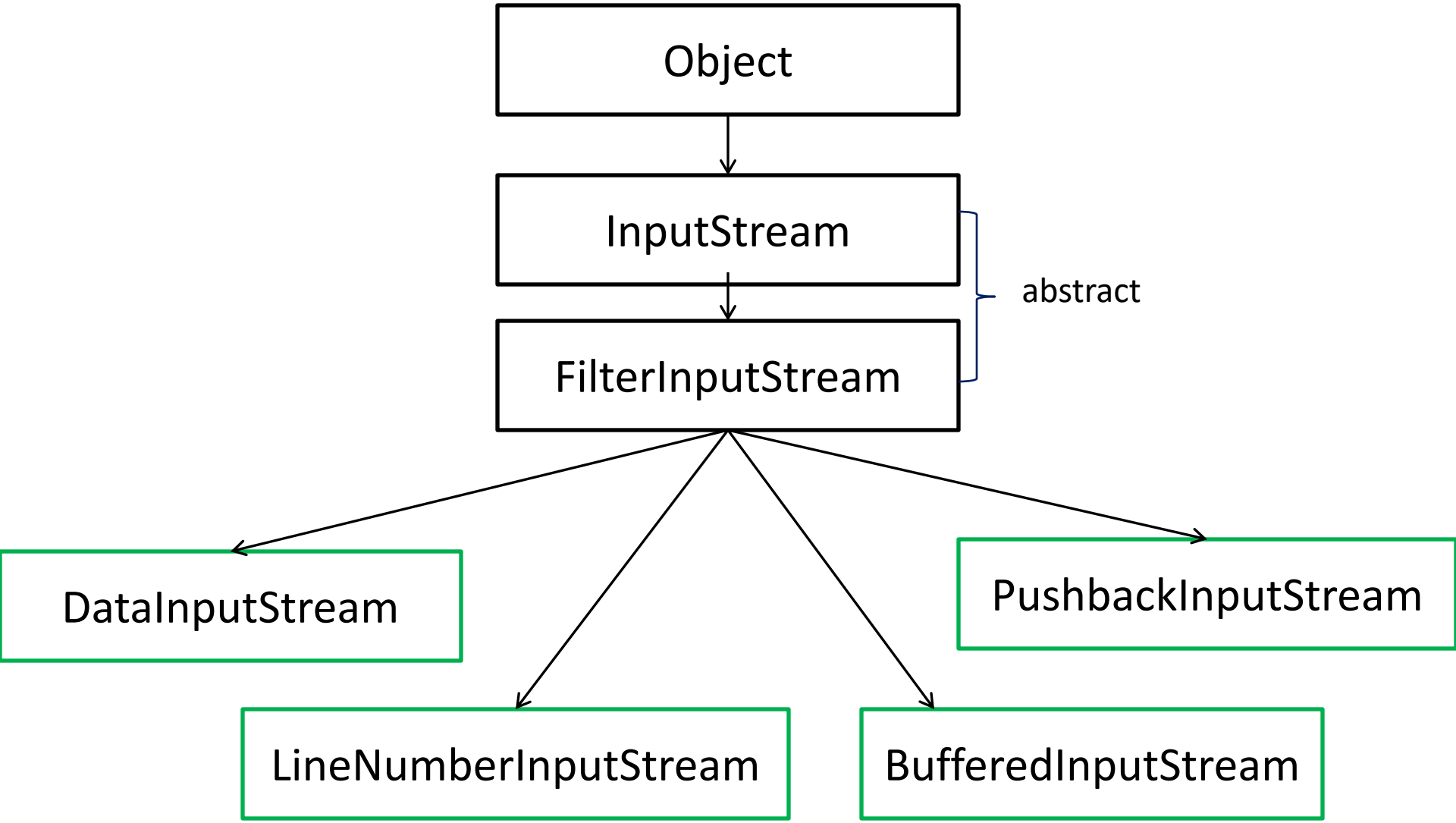
f2.txt - Notepad

File Edit Format View Help

lcom

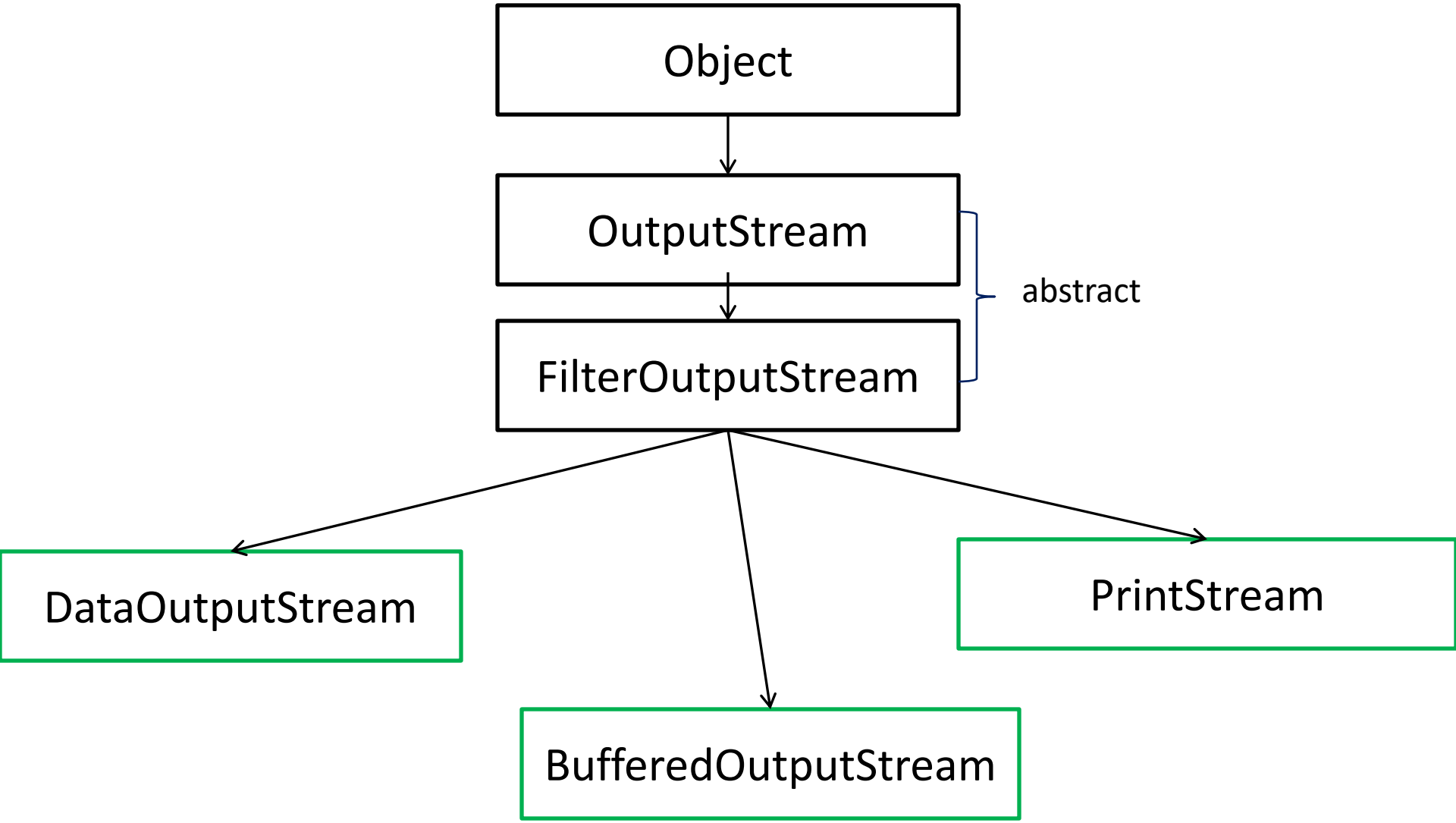
100% Windows (CRLF) UTF-8

FilterStreams



All 4 subclasses of FilterInputStream are called high –level Streams and remaining are called low level streams

FilterStreams



All 3 subclasses of FilterOutputStream are called high –level Streams and remaining are called low level streams

FilterStreams

- **FilterStreams can be linked to another to have high functionality**, but subject to some rules.
- The job of high level streams is to **add extra functionality** to the existing streams.
  - Ex. LineNumberInputStream adds line numbers in the destination file that do not exist in source file.
  - DataInputStream increases performance with readInt() and readLine() methods.
- For high functionality**, one stream can be linked or chained to another**, but obeying some rules.
- Chaining is very simple.
- The output of one stream becomes input to the other (or) pass an object of one stream as parameter to another stream constructor

Rules of chaining

1. The input for a high level stream may come from a low-level stream or another high level stream

   i.e.. **In programming, the constructor of high level stream can be passed with an object of low-level (or) high-level.**

2. Being low-level, the low level stream should work by itself. if not entitled to get passes with any other stream

In programming, the low level streams opens the file and hand it over (passes) to a high level stream .

High level stream cannot open a file directly

High level streams just add extra functionality and depend solely on low-level streams

```
String str="welcome to mit"
StringBufferInputStream sbi=new StringBufferInputStream(str);
LineNumberInputStream lis=new LineNumberInputStream(sbi);
DataInputStream ds=new DataInputStream(lis);
```

Low level

high level

high level

DataInputStream & DataOutputStream

- **These are high-level classes as they are sub classes of FilterInputStream and FilterOutputStream**
- These streams **extra functionality** is that they **can read (or) write primitive Java data types** (integers, doubles)from an underlying input stream in a machine-independent way, instead of byte by byte,
- This increases the performance to some extend, instead of reading & writing byte by byte.
- DataInputStream is **not necessarily safe for multithreaded access**

DataInputStream constructor

- **DataInputStream(InputStream in)**

  This creates a DataInputStream that uses the specified underlying InputStream.

Method & Description

int read(byte[] b)→ This method **reads some number of bytes** from the contained input stream and stores them into the buffer array *b*
int read(byte[] b, int off, int len) → This method reads up to *len* bytes of data from the contained input stream into an array of bytes.
boolean readBoolean() →This method reads one input byte and returns true if that byte is nonzero, false if that byte is zero.

Method & Description

byte readByte() → This method reads and returns one input byte.
char readChar() → This method reads two input bytes and returns a char value.
double readDouble() → This method reads eight input bytes and returns a double value.
float readFloat() → This method reads four input bytes and returns a float value.
void readFully(byte[] b) →This method reads some bytes from an input stream and stores them into the buffer array *b*.
void readFully(byte[] b, int off, int len) →This method reads *len* bytes from an input stream.

int readInt() → This method reads four input bytes and returns an int value.

long readLong() →This method reads eight input bytes and returns a long value.

short readShort() → This method reads two input bytes and returns a short value.

int readUnsignedByte() →This method reads one input byte, zero-extends it to type int, and returns the result, which is therefore in the range 0 through 255.

int readUnsignedShort() → This method reads two input bytes and returns an int value in the range 0 through 65535.

String readUTF() → This method reads in a string that has been encoded using a modified UTF-8 format.

static String readUTF(DataInput in) → This method reads from the stream in a representation of a Unicode character string encoded in modified UTF-8 format; this string of characters is then returned as a String.

int skipBytes(int n) →This method makes an attempt to skip over n bytes of data from the input stream, discarding the skipped bytes.

DataOutputStream

- It lets an application **write primitive java data types** to an output stream in a portable way.
- An application can use a DataInputStream to read the data back in.
- **constructor**

  DataOutputStream(OutputStream out)

  This creates a new data output stream to write data to the specified underlying output stream.

Method & Description

void flush() →This method flushes this data output stream.

int size() →This method returns the current value of the counter written, the number of bytes written to this data output stream so far.

void write(byte[] b, int off, int len) →This method writes len bytes from the specified byte array starting at offset off to the underlying output stream.

void write(int b) → This method writes the specified byte (the low eight bits of the argument b) to the underlying output stream.

void writeBoolean(boolean v) →This method writes a boolean to the underlying output stream as a 1-byte value.

void writeByte(int v) → This method writes out a byte to the underlying output stream as a 1-byte value.

Method & Description

void writeBytes(String s) →This method writes out the string to the underlying output stream as a sequence of bytes.

void writeChar(int v) →This method writes a char to the underlying output stream as a 2-byte value, high byte first.

void writeChars(String s)→This method writes a string to the underlying output stream as a sequence of characters

void writeDouble(double v) →This method converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first.

void writeFloat(float v) →This method converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first.

Method & Description

void writeInt(int v)

This method writes an int to the underlying output stream as four bytes, high byte first.

void writeLong(long v)

This method writes a long to the underlying output stream as eight bytes, high byte first.

void writeShort(int v)

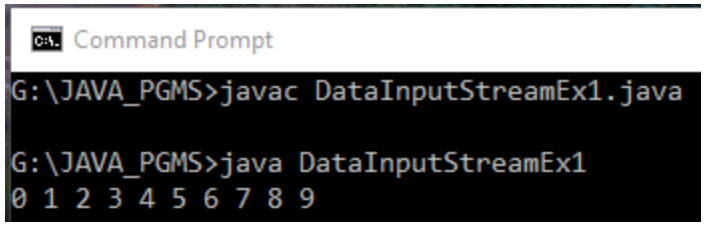
This method writes a short to the underlying output stream as two bytes, high byte first.

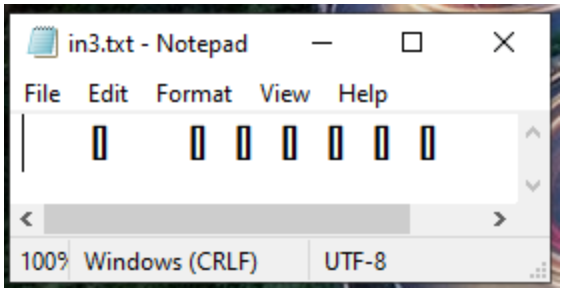
void writeUTF(String str)

This method writes a string to the underlying output stream using modified UTF-8 encoding in a machine-independent manner.

DataInputStream/DataOutputStream Example 1

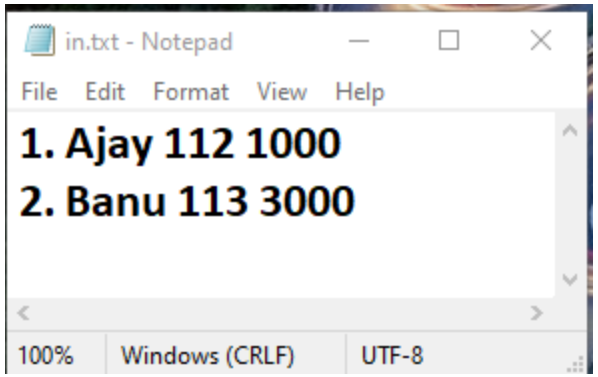
```
import java.io.*;
class DataInputStreamEx1
{
  public static void main(String a[])
  {
     try
     {
          FileOutputStream fout=new FileOutputStream("in3.txt");
          DataOutputStream dsout=new DataOutputStream(fout);
          for(int i=0;i<10;i++)
                    dsout.writeInt(i);
          fout.close();
          FileInputStream fin=new FileInputStream("in3.txt");
          DataInputStream dsin=new DataInputStream(fin);
          int c;
          while(dsin.available()>0)
          {
                    c=dsin.readInt();
                    System.out.print(c+" ");
          }
       }catch(Exception e) {
                    System.out.println("Exception:"+e);
          }
  }
}
```

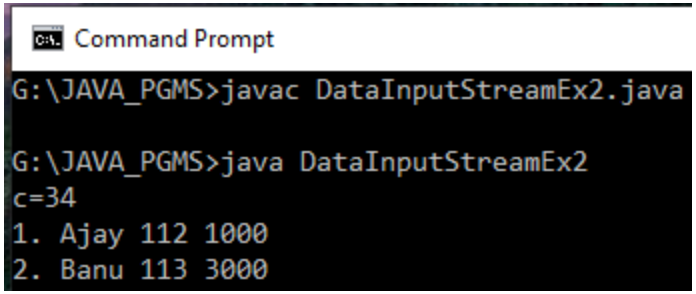




DataInputStream Example2

```
import java.io.*;
class DataInputStreamEx2
{
   public static void main(String a[])
   {
      try
       {
            FileInputStream fin=new FileInputStream("in.txt");
            DataInputStream dsin=new DataInputStream(fin);
            int c=dsin.available();
            System.out.println("c="+c);
            byte b[]=new byte[c];
            dsin.read(b);
            for(int i=0;i<b.length;i++)
            {
                System.out.print((char)b[i]);
            }
        }catch(Exception e)
            {
                        System.out.println("Exception:"+e);
            }
      }
}
```





BufferedInputStream & BufferedOutputStream

- The **functionality of these high level classes is to increase the performance with buffer**
- These streams give an implicit system-defined buffer (generally 2048 bytes) into which data is read and written
- The **buffer decreases the number of transfers between source file context area and destination file context area** and there by performance increases
- The buffer works as a reservoir to store data
- A buffer stands in between an input stream and output stream
- The data is read and put in the buffer instead of sending immediately

BufferedInputStream & BufferedOutputStream

- **When the buffer is full, the buffer is transferred**. This decreases the number of execution control shifting between input and output streams and there by performance increases
- **The size of the buffer allocated depends on the underlying operating system**
- The size of buffer can be requested explicitly using overloaded constructor of BufferedInputStream as follows,

```
BufferedInputStream bis=new BufferedInputStream(fistream,6000)
```

- In the above statement, a buffer of 6000 bytes is allocated by the OS.

Constructor & Description

1. **BufferedInputStream(InputStream in)**

   This creates a BufferedInputStream and saves its argument, the input stream in, for later use.

2. **BufferedInputStream(InputStream in, int size)**

   This creates a BufferedInputStream with the specified buffer size, and saves its argument, the input stream in, for later use.

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.
boolean markSupported()	It tests for the input stream to support the mark

BufferedOutputStream

- It is **used for buffering an output stream**.
- It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written.
- **constructors**

BufferedOutputStream(OutputStream out)

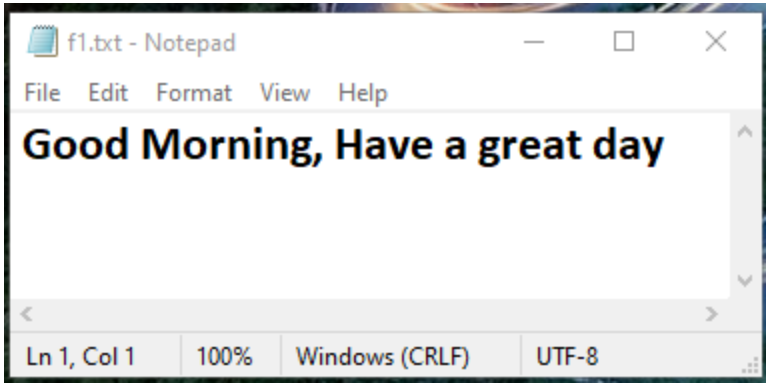
This creates a new buffered output stream to write data to the specified underlying output stream.

BufferedOutputStream(OutputStream out, int size)

This creates a new buffered output stream to write data to the specified underlying output stream with the specified buffer size.

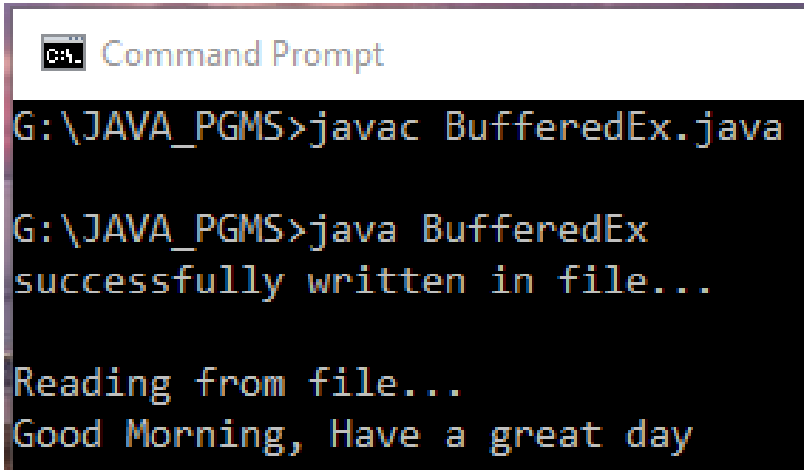
Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream

BufferedInputStream & BufferedOutputStream Example




```
import java.io.*;
class BufferedEx
{
 public static void main(String args[])throws Exception
 {
  try
   {
            FileOutputStream fout=new FileOutputStream("f1.txt");
            BufferedOutputStream bout=new BufferedOutputStream(fout);
            String s="Good Morning, Have a great day";
            byte b[]=s.getBytes();
            bout.write(b);
            bout.flush();
            bout.close();
            fout.close();
            System.out.println("successfully written in file...");
            System.out.println("\nReading from file...");
             FileInputStream fin=new FileInputStream("f1.txt");
             BufferedInputStream bin=new BufferedInputStream(fin);
```

```
 int i;
 while((i=bin.read())!=-1)
 {
   System.out.print((char)i);
 }
  bin.close();
  fin.close();
}catch(Exception e)
  {
    System.out.println("Exception:"+e);
   }
 }
}
```



PushbackInputStream

- Push back is used as an input stream **to allow a byte to be read and then returned(unread) (i.e: "pushed back") to the stream**
- The PushbackInputStream class implements this idea
- It provides a mechanism to "peek" at what is coming from an input stream without disturbing it.
- **Constructor & Description**

PushbackInputStream(InputStream in)

This creates a PushbackInputStream and saves its argument, the input stream in, for later use.

PushbackInputStream(InputStream in, int size)

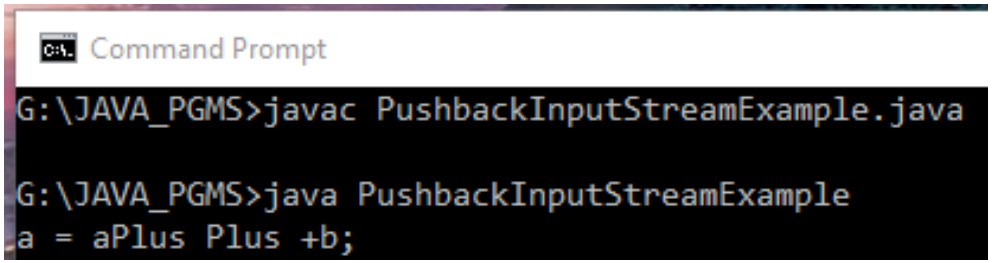
This creates a PushbackInputStream with a pushback buffer of the specified size, and saves its argument, the input stream in, for later use.

Method	Description
int available()	It is used to return the number of bytes that can be read from the input stream.
int read()	It is used to read the next byte of data from the input stream.
boolean markSupported()	It tests if this input stream supports the mark and reset methods, which it does not.
void mark(int readlimit)	It is used to mark the current position in the input stream.
long skip(long x)	It is used to skip over and discard x bytes of data.
void unread(int b)	It is used to pushes back the byte by copying it to the pushback buffer.
void unread(byte[] b)	It is used to pushes back the array of byte by copying it to the pushback buffer.
void reset()	It is used to reset the input stream.
void close()	It is used to close the input stream.

PushbackInputStream Example

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.PushbackInputStream;
public class PushbackInputStreamExample
{
      public static void main(String[] args)
     {
            String strExpression = "a = a++ + b;";
             byte bytes[] = strExpression.getBytes();
             ByteArrayInputStream bis = new ByteArrayInputStream(bytes);
             PushbackInputStream pis = new PushbackInputStream(bis);
             int ch,c;
             try
             {
                while( (ch = pis.read())!= -1)
                 {
                      if(ch == '+')
                       {
                         if( (ch = pis.read()) == '+')
                             {
                                   System.out.print("Plus Plus");
                             }
```

```
                else
                    {
                        System.out.print("+");
                    }
                }else
                {
                  System.out.print((char)ch);
                }
            }
        }
        catch(IOException ioe)
        {
            System.out.println("Exception while reading" + ioe);
        }
    }
}
```



PrintStream class

- This class **prints java primitive values and object to a stream as text**
- The **PrintStream class automatically flushes the data so there is no need to call flush() method**.
- None of the methods in this class throws an Exception

Constructor & Description

PrintStream(File file)

This creates a new print stream, without automatic line flushing, with the specified file

PrintStream(File file, String csn)

This creates a new print stream, without automatic line flushing, with the specified file and charset.

PrintStream(OutputStream out)

This creates a new print stream.

PrintStream(OutputStream out, boolean autoFlush)

This creates a new print stream.

PrintStream(OutputStream out, boolean autoFlush, String encoding)

This creates a new print stream.

PrintStream(String fileName)

This creates a new print stream, without automatic line flushing, with the specified file name.

PrintStream(String fileName, String csn)

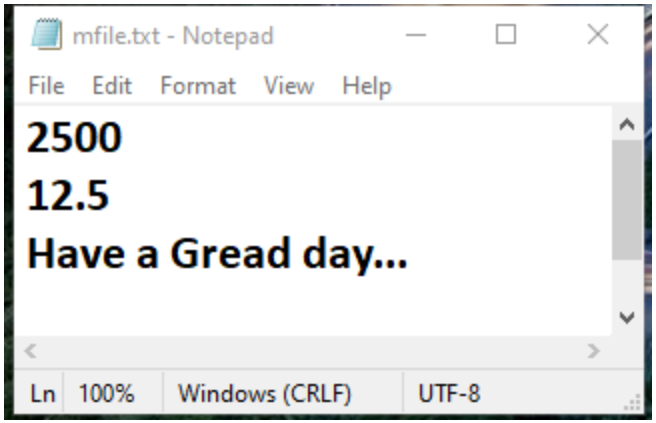
This creates a new print stream, without automatic line flushing, with the specified file name and charset.

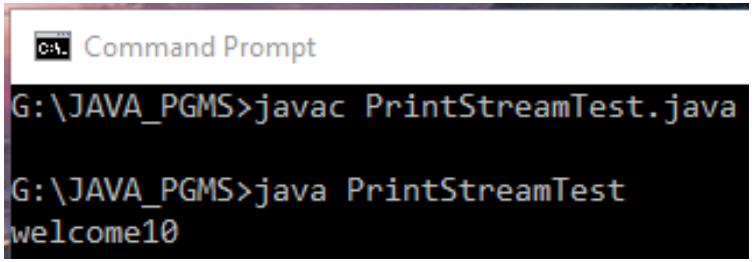
Method	Description
void print(boolean b)	It prints the specified boolean value.
void print(char c)	It prints the specified char value.
void print(char[] c)	It prints the specified character array values.
void print(int i)	It prints the specified int value.
void print(long l)	It prints the specified long value.
void print(float f)	It prints the specified float value.
void print(double d)	It prints the specified double value.
void print(String s)	It prints the specified string value.
void print(Object obj)	It prints the specified object value.

void println(boolean b)	It prints the specified boolean value and terminates the line.
void println(char c)	It prints the specified char value and terminates the line.
void println(char[] c)	It prints the specified character array values and terminates the line.
void println(int i)	It prints the specified int value and terminates the line.
void println(long l)	It prints the specified long value and terminates the line.
void println(float f)	It prints the specified float value and terminates the line.
void println(double d)	It prints the specified double value and terminates the line.
void println(String s)	It prints the specified string value and terminates the line.
void println(Object obj)	It prints the specified object value and terminates the line.

PrintStreamExample

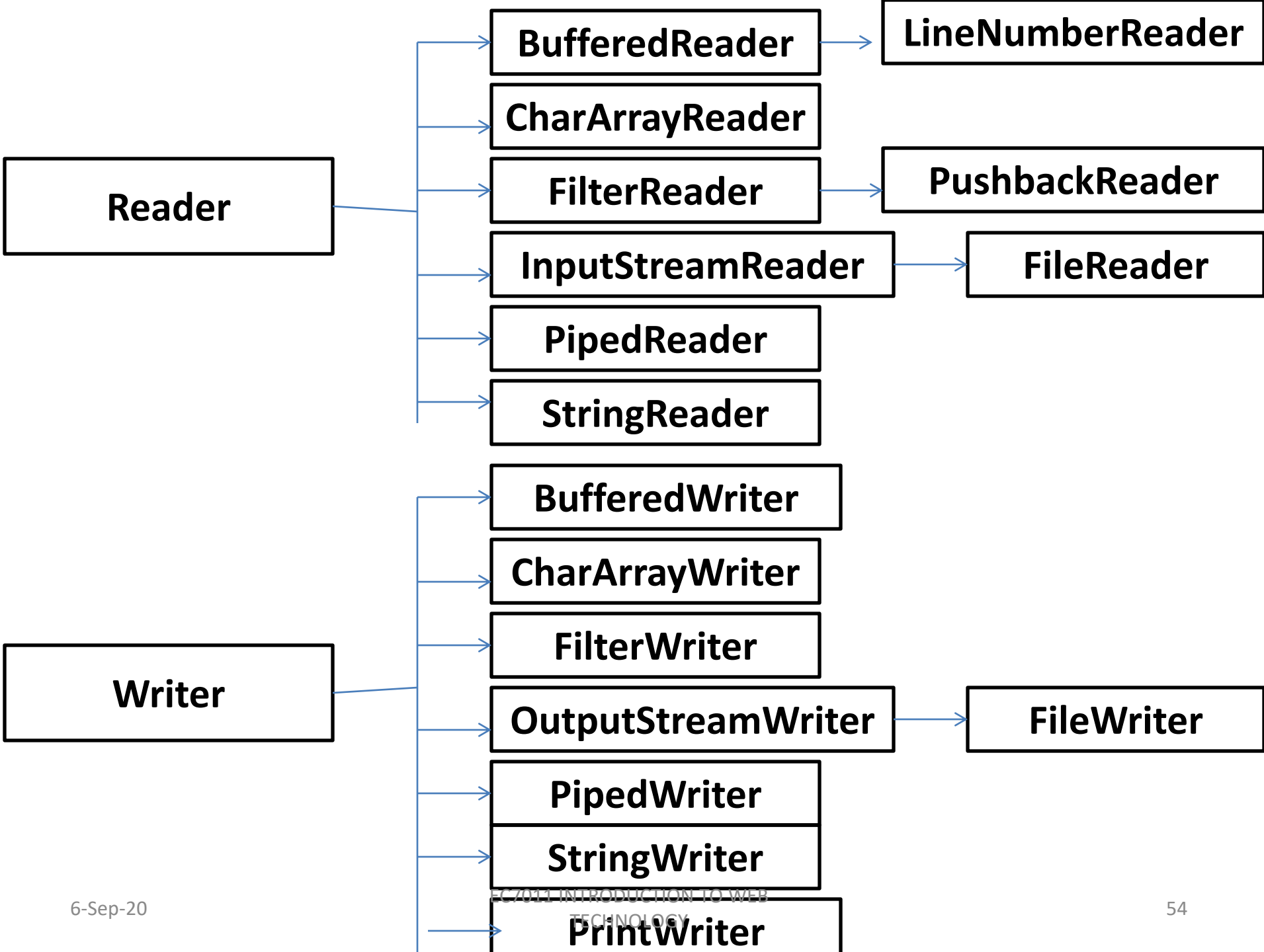
```
import java.io.*;
class PrintStreamTest
{
 public static void main(String args[])throws Exception
 {
  FileOutputStream fout=new FileOutputStream("mfile.txt");
  PrintStream pout=new PrintStream(fout);
  pout.println(2500);
  pout.println(12.5);
  pout.println("Have a Gread day...");
  pout.close();
  fout.close();
  pout=new PrintStream(System.out);
            pout.print("welcome");
            int c=10;
            pout.print(c);
            pout.close();
 }
}
```





Reader & Writer classes

- The **Java Reader** and **Java Writer** class in Java **IO work much like the InputStream and OutputStreamwith** the exception that **Reader and Writer are character based**.
- Input streams read bytes whereas readers are able to read characters
- These are intended for reading and writing text.


Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
StringWriter
PrintWriter

FileWriter & FileReader

- These classes are used to write and read data from text files
- These are character oriented classes used for file handling in java
- Java have suggested not to use the FileInputStream and FileOutputStream classes if we have to read and write the textual information

FileWriter Class

- Java FileWriter class is used to **write character-oriented data to a file**. It is character-oriented class which is used for file handling in java.
- Unlike FileOutputStream class, you don't need to convert string into byte array because it provides method to write string directly.
- **Constructors**

1. FileWriter(String file)

   Creates a new file. It gets file name in string.

2. 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.

FileReader Class

- Java FileReader class 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.
- **Constructors**

1.FileReader(String file)

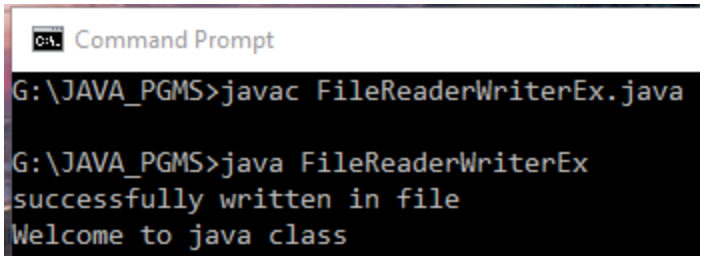
It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException

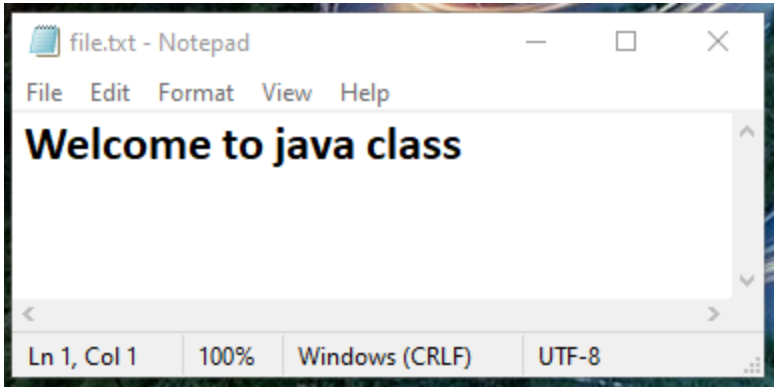
2. FileReader(File file)It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

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.

FileWriter & FileReader Example

```
import java.io.*;
class FileReaderWriterEx
{
 public static void main(String args[])
 {
   try
   {
            FileWriter fw=new FileWriter("file.txt");
            fw.write("Welcome to java class");
            fw.close();
            System.out.println("successfully written in file");
            FileReader fr=new FileReader("file.txt");
            int i;
            while((i=fr.read())!=-1)
                System.out.print((char)i);
            fr.close();
     }catch(Exception e)
      {
            System.out.println(e);
      }
  }
}
```





CharArrayReader

- CharArrayReader implements a **character buffer** that can be used as character-input stream
- It is **similar to ByteArrayInputStream**
- **constructors**

1. **CharArrayReader(char[] buf)**

   This creates a CharArrayReader from the specified array of chars.

2. **CharArrayReader(char[] buf, int offset, int length)**

   This creates a CharArrayReader from the specified array of chars.

Method	Description
int read()	It is used to read a single character
int read(char[] b, int off, int len)	It is used to read characters into the portion of an array.
boolean ready()	It is used to tell whether the stream is ready to read.
boolean markSupported()	It is used to tell whether the stream supports mark() operation.
long skip(long n)	It is used to skip the character in the input stream.
void mark(int readAheadLimit)	It is used to mark the present position in the stream.
void reset()	It is used to reset the stream to a most recent mark.
void close()	It is used to closes the stream.

CharArrayWriter

- The CharArrayWriter class can be used **to write common data to multiple files**.
- This class inherits Writer class.
- Its **buffer automatically grows** when data is written in this stream.
- Calling the close() method on this object has no effect.
- **constructors**

1.CharArrayWriter()

This creates a CharArrayReader from the specified array of chars.

2.**CharArrayWriter(int initialSize)**

This creates a new CharArrayWriter with the specified initial size.

Method	Description
int size()	It is used to return the current size of the buffer.
char[] toCharArray()	It is used to return the copy of an input data.
String toString()	It is used for converting an input data to a string.
CharArrayWriter append(char c)	It is used to append the specified character to the writer.
CharArrayWriter append(CharSequence csq)	It is used to append the specified character sequence to the writer.
CharArrayWriter append(CharSequence csq, int start, int end)	It is used to append the subsequence of a specified character to the writer.
void write(int c)	It is used to write a character to the buffer.
void write(char[] c, int off, int len)	It is used to write a character to the buffer.
void write(String str, int off, int len)	It is used to write a portion of string to the buffer.
void writeTo(Writer out)	It is used to write the content of buffer to different character stream.
void flush()	It is used to flush the stream.
void reset()	It is used to reset the buffer.
void close()	It is used to close the stream.

CharArrayReader and Writer Example

```
import java.io.*;
public class CharArrayReaderWriterEx
{
  public static void main(String args[])
  {
    try
    {
            char c[] = {'w','e','l','c','o','m','e','t','o','M','I','T'};
            CharArrayReader r1 = new CharArrayReader(c);
            CharArrayReader r2 = new CharArrayReader(c, 2, 5);
            int i;
            while((i = r1.read()) != -1)
             {
                System.out.print((char)i);
            }
            System.out.println();
            while((i = r2.read()) != -1)
            {
                System.out.print((char)i);
            }
```

```
CharArrayWriter out = new CharArrayWriter();
 out.write(c);
 FileWriter f1 = new FileWriter("o1.txt");
out.writeTo(f1); //File written successfully.

FileWriter f2 = new FileWriter("o2.txt");
out.writeTo(f2); //File written successfully.

System.out.println("\nBuffer as a string");
System.out.println(out.toString());
System.out.println("\nInto array");
char ch[] = out.toCharArray();
for (i=0; i<ch.length; i++)
 {
            System.out.print(ch[i]);
}
f1.close();
f2.close();
//CharArrayWriter is closed.
out.close();
```

```
FileWriter f3 = new FileWriter("o3.txt");
//Write again to a file. No Exception from CharArrayWriter but no data will be written.
out.writeTo(f3);
 System.out.println("\nFile written successfully.");
}catch(Exception e)
  {
            System.out.println("Exception :"+e);
   }
  }
}
```

Command Prompt

```
G:\JAVA_PGMS>javac CharArrayReaderWriterEx.java

G:\JAVA_PGMS>java CharArrayReaderWriterEx
welcometoMIT
lcome
Buffer as a string
welcometoMIT

Into array
welcometoMIT
File written successfully.
```

o1.txt - Notepad

File Edit Format View Help

welcometoMIT

Ln 100% Windows (CRLF) UTF-8

o2.txt - Notepad

File Edit Format View Help

welcometoMIT

Ln 100% Windows (CRLF) UTF-8

o3.txt - Notepad

File Edit Format View Help

Ln 100% Windows (CRLF) UTF-8

StringReader

- **It is a character stream with string as a source**.
- It takes an input string and changes it into character stream.
- It inherits Reader class.
- In StringReader class, system resources like network sockets and files are not used, therefore closing the StringReader is not necessary.
- constructor

  StringReader(String s)

  This creates a new string reader.

Method	Description
int read()	It is used to read a single character.
int read(char[] cbuf, int off, int len)	It is used to read a character into a portion of an array.
boolean ready()	It is used to tell whether the stream is ready to be read.
boolean markSupported()	It is used to tell whether the stream support mark() operation.
long skip(long ns)	It is used to skip the specified number of character in a stream
void mark(int readAheadLimit)	It is used to mark the mark the present position in a stream.
void reset()	It is used to reset the stream.
void close()	It is used to close the stream.

StringWriter

- It is a character stream that **collects output from string buffer**, which can be used to construct a string. The StringWriter class inherits the Writer class.
- In StringWriter class, system resources like network sockets and files are not used, therefore closing the StringWriter is not necessary.
- **constructors**

1. StringWriter()

This creates a new string writer using the default initial string-buffer size.

2.**StringWriter(int initialSize)**

This creates a new string writer using the specified initial string-buffer size.

Method	Description
void write(int c)	It is used to write the single character.
void write(String str)	It is used to write the string.
void write(String str, int off, int len)	It is used to write the portion of a string.
void write(char[] cbuf, int off, int len)	It is used to write the portion of an array of characters.
String toString()	It is used to return the buffer current value as a string.
StringBuffer getBuffer()	It is used t return the string buffer.
StringWriter append(char c)	It is used to append the specified character to the writer.
StringWriter append(CharSequence csq)	It is used to append the specified character sequence to the writer.
StringWriter append(CharSequence csq, int start, int end)	It is used to append the subsequence of specified character sequence to the writer.
void flush()	It is used to flush the stream.
void close()	It is used to close the stream.

StringReader & Writer Example

```
import java.io.*;
public class StringReaderWriterEx
{
   public static void main(String[] args)
   {
      String str = "Good Morning! \nWelcome to MIT.";
      StringReader sr = new StringReader(str);
      int i=0;
      try
      {
            while((i=sr.read())!=-1)
            {
               System.out.print((char)i);
            }
            String s = "Hello";
            // create a new writer
            StringWriter sw = new StringWriter();
            // write strings
            sw.write(s);
            sw.write(" World");
            // print result by converting to string
            System.out.println(sw.toString());
      } catch (IOException e)    {  System.out.println("Exception:"+e);}
      }
}
```

Command Prompt

```
G:\JAVA_PGMS>javac StringReaderWriterEx.java

G:\JAVA_PGMS>java StringReaderWriterEx
Good Morning!
Welcome to MIT.Hello World
```

InputStreamReader

- It is a **bridge from byte streams to character streams**.
- It **reads bytes and decodes them into characters** using a specified charset.
- **constructors**

1. **InputStreamReader(InputStream in)**

   This creates an InputStreamReader that uses the default charset.

2. **InputStreamReader(InputStream in, Charset cs)**

   This creates an InputStreamReader that uses the given charset.

3. **InputStreamReader(InputStream in, CharsetDecoder dec)**

   This creates an InputStreamReader that uses the given charset decoder.

4. **InputStreamReader(InputStream in, String charsetName)**

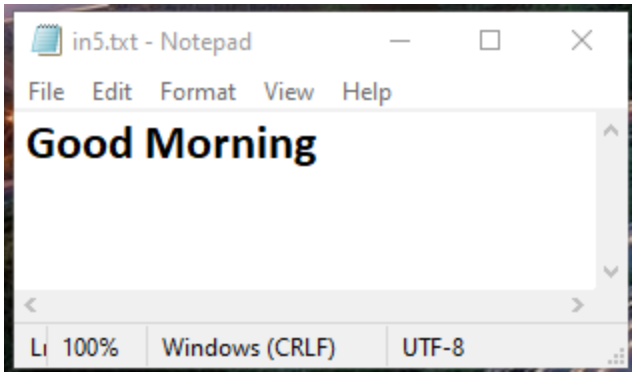
   This creates an InputStreamReader that uses the named charset.

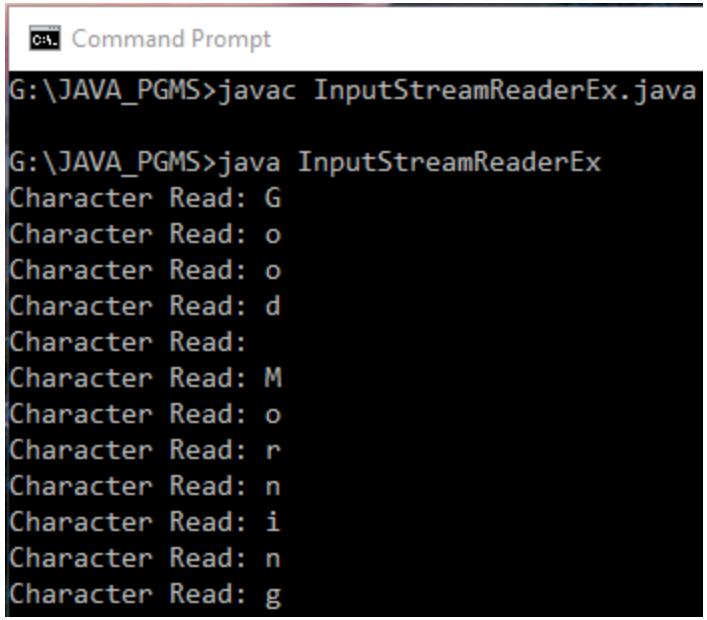
Method & Description

1. **void close()→**This method closes the stream and releases any system resources associated with it.
2. **String getEncoding()→**This method returns the name of the character encoding being used by this stream.
3. **int read() →**This method reads a single character.
4. **int read(char[] cbuf, int offset, int length)→** This method reads characters into a portion of an array.
5. **boolean ready()→** This method tells whether this stream is ready to be read.

InputStreamReader Example

```
import java.io.*;
public class InputStreamReaderEx
 {
  public static void main(String[] args)
  {
    char c;
    int i;
    try
    {
      FileInputStream fis = new FileInputStream("in5.txt");
      InputStreamReader isr = new InputStreamReader(fis);
      while((i=isr.read())!=-1)
      {
        // int to character
        c=(char)i;
        // print char
        System.out.println("Character Read: "+c);
       }
      fis.close();
      isr.close();
     } catch (Exception e) { e.printStackTrace(); }
  }
}
```





BufferedReader

- It is used **to read the text from a character-based input stream.**
- It can be used to **read data line by line by readLine()** method.
- It makes the performance fast.
- It inherits Reader class.
- **constructors**

1. **BufferedReader(Reader rd)**

   It is used to create a buffered character input stream that uses the default size for an input buffer.

2. **BufferedReader(Reader rd, int size)**

   It is used to create a buffered character input stream that uses the specified size for an input buffer.

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.

BufferedWriter

- It is used to **provide buffering for Writer instances**.
- It makes the performance fast.
- It inherits Writer class.
- The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
- **constructors**

1. **BufferedWriter(Writer wrt)**

   It is used to create a buffered character output stream that uses the default size for an output buffer.

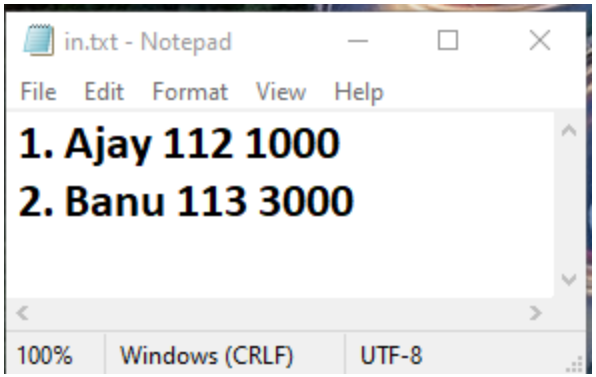
2. **BufferedWriter(Writer wrt, int size)**

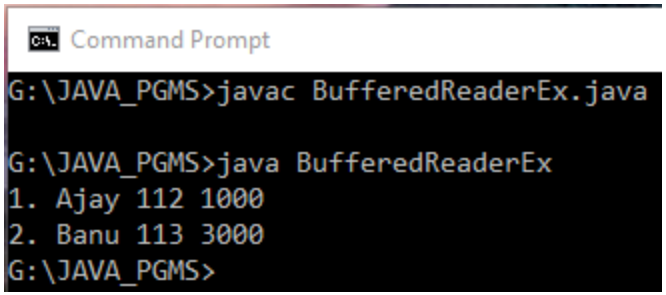
   It is used to create a buffered character output stream that uses the specified size for an output buffer.

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

BufferedReader Example

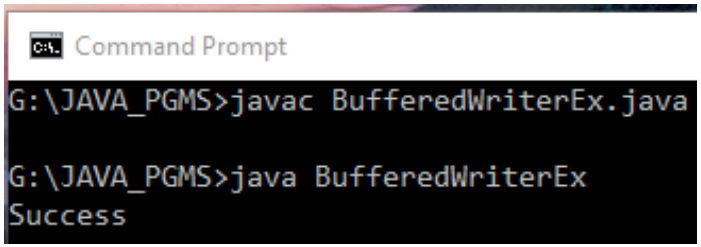
```
import java.io.*;
public class BufferedReaderEx
 {
   public static void main(String args[])
   {
       try
      {
            FileReader fr=new FileReader("in.txt");
            BufferedReader br=new BufferedReader(fr);
            /*int i;
            while((i=br.read())!=-1)
            {
                  System.out.print((char)i);
            }*/
            String str;
            while((str=br.readLine())!=null)
            {
                  System.out.println(str);
            }
            br.close();
            fr.close();
       }catch(Exception e){System.out.println("Exception:"+e);}
  }
}
```

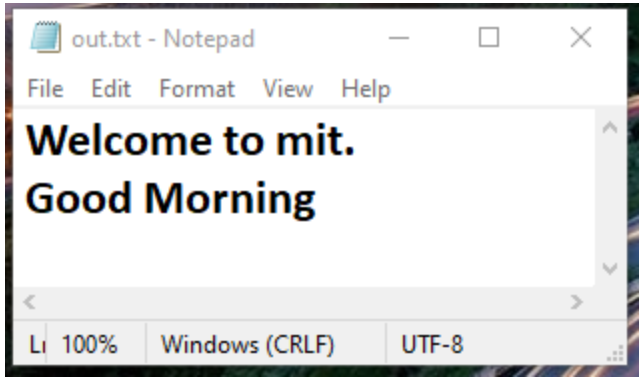




BufferedWriter Example

```
import java.io.*;
public class BufferedWriterEx
 {
   public static void main(String[] args)
   {
     try
     {
            FileWriter writer = new FileWriter("out.txt");
            BufferedWriter buffer = new BufferedWriter(writer);
            buffer.write("Welcome to mit.");
            buffer.newLine();
            buffer.write("Good Morning");
            buffer.close();
            System.out.println("Success");
      }catch(Exception e)
            {
              System.out.println("Exception:"+e);
            }
   }
}
```





FilterStreams (character)

- These are concrete classes that includes some filtering capabilities as data is read (or) written by another stream
- Eg. FilterReader object takes input from another reader object and does processing for extra functionality and returns the processed data.

FilterReader

- It is an abstract class of character streams with filter capabilities on reading side
- Its equivalent in byte streams is FilterInputStream
- It includes only one sub class, PushbackReader
- Using PushbackReader is similar to PushbackInputStream

FilterWriter

- It is an abstract class for writing character data
- Its equivalent in byte streams is FilterOutputStream
- It does not have any sub class

PushBackReader Example

```
import java.io.*;
public class PushBackReaderEx
{
 public static void main(String args[]) throws IOException
 {
  String str="welcome";
  StringReader sr=new StringReader(str);
  PushbackReader pr = new PushbackReader(sr);
  int temp;
  // read first character and print
  temp = pr.read();
  System.out.println("The first character: " + (char) temp);
  // read the second character and print
  temp = pr.read();
  System.out.println("The second character: " + (char) temp);
  // read the third character and print
  temp = pr.read();
  System.out.println("The third character: " + (char) temp);
   pr.unread(temp);        // unread the third character
    temp = pr.read();      // read it again and print
    System.out.println("Reading again third character: " + (char) temp);
 }
}
```

Command Prompt

```
G:\JAVA_PGMS>javac PushBackReaderEx.java

G:\JAVA_PGMS>java PushBackReaderEx
The first character: w
The second character: e
The third character: l
Reading again third character: l
```