Binary I/O

Motivation

Data stored in a text file is represented in human-readable form. Data stored in a binary file is represented in binary form.

You cannot read binary files. They are designed to be read by programs. For example, Java source programs are stored in text files and can be read by a text editor, but Java classes are stored in binary files and are read by the JVM.

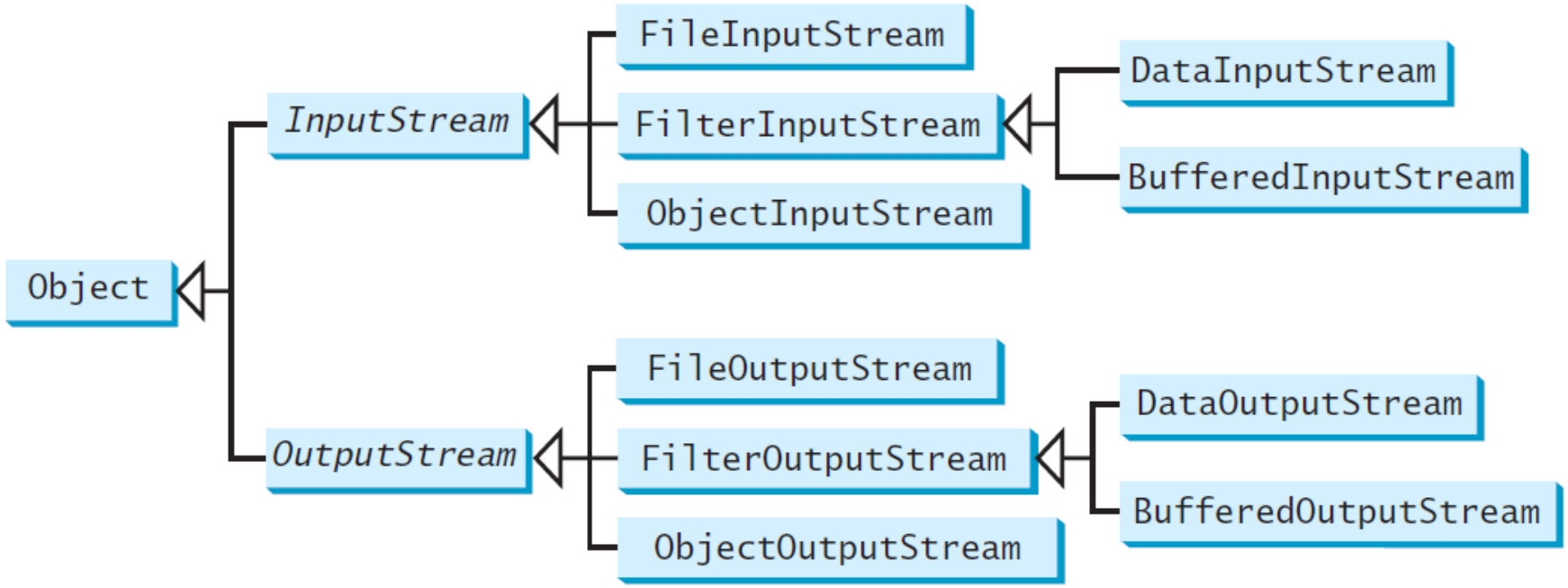
The advantage of binary files is that they are more efficient to process than text files.

Text File vs. Binary File

- Data stored in a text file are represented in human-readable form. Data stored in a binary file are represented in binary form. You cannot read binary files. Binary files are designed to be read by programs. For example, the Java source programs are stored in text files and can be read by a text editor, but the Java classes are stored in binary files and are read by the JVM. The advantage of binary files is that they are more efficient to process than text files.

- Although it is not technically precise and correct, you can imagine that a text file consists of a sequence of characters and a binary file consists of a sequence of bits. For example, the decimal integer 199 is stored as the sequence of three characters: '1', '9', '9' in a text file and the same integer is stored as a byte-type value C7 in a binary file, because decimal 199 equals to hex C7.

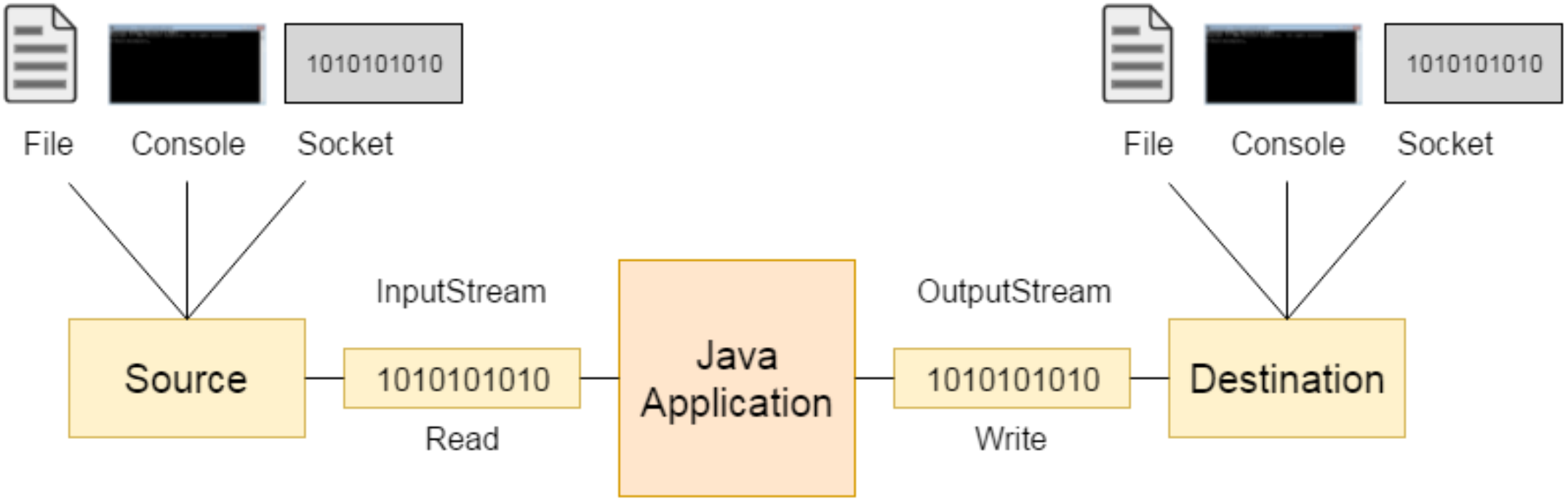
Binary I/O Classes



OutputStream vs InputStream

- The explanation of OutputStream and InputStream classes are given below:
- OutputStream
- Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
- 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 vs InputStream



OutputStream class

- OutputStream class is an abstract class. 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.
- Useful methods of OutputStream

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

InputStream class

- InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.
- Useful methods of InputStream

Method	Description
1) 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.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

Java FileOutputStream Class

- Java FileOutputStream 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.
- FileOutputStream class methods

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.

Java FileOutputStream Example 1: write byte

```
1.import java.io.FileOutputStream;
2.public class FileOutputStreamExample {
3.    public static void main(String args[]){
4.          try{
5.             FileOutputStream fout=new FileOutputStream("D:\\testout.t
xt");
6.            fout.write(65);
7.            fout.close();
8.            System.out.println("success...");
9.           }catch(Exception e){System.out.println(e);}
10.     }
11.}
```

Output:
success….

The content of a text file **testout.txt** is set with the data **A**.

Java FileOutputStream example 2: write string

```
1.import java.io.FileOutputStream;
2.public class FileOutputStreamExample {
3.    public static void main(String args[]){
4.          try{
5.            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.            String s="Welcome to NSU";
7.            byte b[]=s.getBytes();//converting string into byte array
8.            fout.write(b);
9.            fout.close();
10.             System.out.println("success...");
11.            }catch(Exception e){System.out.println(e);}
12.     }
13.}
```

Output:
success….

The content of a text file **testout.txt** is set with the data **Welcome to NSU**

Java FileInputStream Class

- Java FileInputStream 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. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.
- Jav

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 closes the stream.

Java FileInputStream example 1: read single character

```
1.import java.io.FileInputStream;
2.public class DataStreamExample {
3.     public static void main(String args[]){
4.          try{
5.            FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.            int i=fin.read();
7.            System.out.print((char)i);
8.
9.            fin.close();
10.           }catch(Exception e){System.out.println(e);}
11.         }
12.        }
```

Note: Before running the code, a text file named as **"testout.txt"** is required to be created. In this file, we are having following content:

Output:

As we have already a text file **testout.txt** with the content **Welcome to NSU,** After executing the above program, you will get a single character from the file which is 87 (in byte form). To see the text, you need to convert it into character. It will be **"W"**

Java FileInputStream example 2: read all characters

```
1.import java.io.FileInputStream;
2.public class DataStreamExample {
3.     public static void main(String args[]){
4.          try{
5.            FileInputStream fin=new FileInputStream("D:\\testout.txt");

6.            int i=0;
7.            while((i=fin.read())!=-1){
8.             System.out.print((char)i);
9.            }
10.             fin.close();
11.           }catch(Exception e){System.out.println(e);}
12.         }
13.       }
```

Output:
Welcome to NSU

Java BufferedOutputStream Class

- Java BufferedOutputStream class 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.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class.
- Java BufferedOutputStream class constructors :

Constructor	Description
BufferedOutputStream(OutputStream os)	It creates the new buffered output stream which is used for writing the data to the specified output stream.
BufferedOutputStream(OutputStream os, int size)	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

- Java BufferedOutputStream class methods :

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.

Example of BufferedOutputStream class:

- In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object.
- The **flush()** flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.

Example of BufferedOutputStream class:

```
1.import java.io.*;
2.public class BufferedOutputStreamExample{
3.public static void main(String args[])throws Exception{
4.     FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
5.     BufferedOutputStream bout=new BufferedOutputStream(fout);
6.     String s="Welcome to NSU";
7.     byte b[]=s.getBytes();
8.     bout.write(b);
9.     bout.flush();
10.      bout.close();
11.      fout.close();
12.      System.out.println("success");
```

Output:
success….

The content of a text file **testout.txt** is set with the data **Welcome to NSU**

Java BufferedInputStream Class

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:

  - When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.

  - When a BufferedInputStream is created, an internal buffer array is created.

Java BufferedInputStream Class

- Java BufferedInputStream class constructors

Constructor	Description
BufferedInputStream(InputStream IS)	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
BufferedInputStream(InputStream IS, int size)	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

- Java BufferedInputStream class methods

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.

Example of Java BufferedInputStream

```
1.import java.io.*;
2.public class BufferedInputStreamExample{
3. public static void main(String args[]){
4.  try{
5.    FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.    BufferedInputStream bin=new BufferedInputStream(fin);
7.    int i;
8.    while((i=bin.read())!=-1){
9.     System.out.print((char)i);
10.    }
11.    bin.close();
12.    fin.close();
13.  }catch(Exception e){System.out.println(e);}
14. }
15.}
```

Here, we are assuming that you have following data in **"testout.txt"** file:
Welcome to NSU

Output:
Welcome to NSU

Java SequenceInputStream Class

- Java SequenceInputStream class is used to read data from multiple streams. It reads data sequentially (one by one).
- Constructors of SequenceInputStream class:

Constructor	Description
SequenceInputStream(InputStream s1, InputStream s2)	creates a new input stream by reading the data of two input stream in order, first s1 and then s2.
SequenceInputStream(Enumeration e)	creates a new input stream by reading the data of an enumeration whose type is InputStream.

- Methods of SequenceInputStream class

Method	Description
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 len bytes of data from the input stream into the array of bytes.
int available()	It is used to return the maximum number of byte that can be read from an input stream.
void close()	It is used to close the input stream.

Java SequenceInputStream Example

```
1.import java.io.*;
2.class InputStreamExample {
3.  public static void main(String args[])throws Exception{
4.   FileInputStream input1=new FileInputStream("D:\\testin.txt");
5.   FileInputStream input2=new FileInputStream("D:\\testout.txt");
6.   SequenceInputStream inst=new SequenceInputStream(input1, input2);
7.   int j;
8.   while((j=inst.read())!=-1){
9.    System.out.print((char)j);
10.   }
11.   inst.close();
12.   input1.close();
13.   input2.close();
14.  }
15.}
```

we are assuming that you have two files: testin.txt and testout.txt which have following information:

testin.txt:
Welcome to Java IO Programming.
testout.txt:
It is the example of Java SequenceInputStream class.
After executing the program, you will get following output:
Output:
Welcome to Java IO Programming. It is the example of Java SequenceInputStream class.

Example that reads the data from two files and writes into another file

```
1.import java.io.*;
2.class Input1{
3.  public static void main(String args[])throws Exception{
4.   FileInputStream fin1=new FileInputStream("D:\\testin1.txt");
5.   FileInputStream fin2=new FileInputStream("D:\\testin2.txt");
6.   FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
7.   SequenceInputStream sis=new SequenceInputStream(fin1,fin2);
8.   int i;
9.   while((i=sis.read())!=-1)
10.   {
11.     fout.write(i);
12.   }
13.   sis.close();
14.   fout.close();
15.   fin1.close();
16.   fin2.close();
17.   System.out.println("Success..");
18.  }
19.}
```

Output:
Succeess...
testout.txt:
Welcome to Java IO Programming. It is the example of Java SequenceInputStream **class**.

Java ByteArrayOutputStream Class

- Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.
- The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.
- The buffer of ByteArrayOutputStream automatically grows according to data.
- Java ByteArrayOutputStream class constructors:

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

Java ByteArrayOutputStream Class

- Java ByteArrayOutputStream class methods

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.

Example of Java ByteArrayOutputStream:

ByteArrayOutputStream class to write common data into 2 files: f1.txt and f2.txt.

```
1.import java.io.*;
2.public class DataStreamExample {
3.public static void main(String args[])throws Exception{
4.      FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");
5.      FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");
6.      ByteArrayOutputStream bout=new ByteArrayOutputStream();
7.      bout.write(65);
8.      bout.writeTo(fout1);
9.      bout.writeTo(fout2);
10.       bout.flush();
11.       bout.close();//has no effect
12.       System.out.println("Success...");
13.     }
14.    }
```

```
Output:
Success...
f1.txt:
A
f2.txt:
A
```

Java ByteArrayInputStream Class

- The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.
- Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.
- The buffer of ByteArrayInputStream automatically grows according to data.
- Java ByteArrayInputStream class constructors:

Constructor	Description
ByteArrayInputStream(byte[] ary)	Creates a new byte array input stream which uses **ary** as its buffer array.
ByteArrayInputStream(byte[] ary, int offset, int len)	Creates a new byte array input stream which uses **ary** as its buffer array that can read up to specified **len** bytes of data from an array.

Java ByteArrayInputStream Class

- Java ByteArrayInputStream class methods

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.

Example of Java ByteArrayInputStream

- ByteArrayInputStream class to read byte array as input stream.

```
1.import java.io.*;
2.public class ReadExample {
3.  public static void main(String[] args) throws IOException {
4.    byte[] buf = { 35, 36, 37, 38 };
5.    // Create the new byte array input stream
6.    ByteArrayInputStream byt = new ByteArrayInputStream(buf);
7.    int k = 0;
8.    while ((k = byt.read()) != -1) {
9.      //Conversion of a byte into character
10.       char ch = (char) k;
11.       System.out.println("ASCII value of Character is:" + k + "; Special character is:
" + ch);
12.    }
13.  }
14.}
```

Output:

```
ASCII value of Character is:35; Special character is: #
ASCII value of Character is:36; Special character is: $
ASCII value of Character is:37; Special character is: %
ASCII value of Character is:38; Special character is: &
```

Java DataOutputStream Class

- Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.
- Java application generally uses the data output stream to write data that can later be read by a data input stream.

Java DataOutputStream Class

Java DataOutputStream class methods :

Method	Description
int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream.
void writeBoolean(boolean v)	It is used to write Boolean to the output stream as a 1-byte value.
void writeChar(int v)	It is used to write char to the output stream as a 2-byte value.
void writeChars(String s)	It is used to write string to the output stream as a sequence of characters.
void writeByte(int v)	It is used to write a byte to the output stream as a 1-byte value.
void writeBytes(String s)	It is used to write string to the output stream as a sequence of bytes.
void writeInt(int v)	It is used to write an int to the output stream
void writeShort(int v)	It is used to write a short to the output stream.
void writeShort(int v)	It is used to write a short to the output stream.
void writeLong(long v)	It is used to write a long to the output stream.
void writeUTF(String str)	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
void flush()	It is used to flushes the data output stream.

Example of DataOutputStream class

```
1.import java.io.*;
2.public class OutputExample {
3.    public static void main(String[] args) throws IOException {
4.        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
5.        DataOutputStream data = new DataOutputStream(file);
6.        data.writeInt(65);
7.        data.flush();
8.        data.close();
9.        System.out.println("Succcess...");
10.    }
11.}
```

```
Output:
Success...
testout.txt:
A
```

Java DataInputStream Class

- Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.
- Java application generally uses the data output stream to write data that can later be read by a data input stream.

Method	Description
int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	It is used to skip over x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.
void readFully(byte[] b)	It is used to read bytes from the input stream and store them into the buffer array.
void readFully(byte[] b, int off, int len)	It is used to read **len** bytes from the input stream

Example of DataInputStream class

- In this example, we are reading the data from the file testout.txt file.

```
1.package com.javatpoint;
2.import java.io.*;
3.public class DataStreamExample {
4.  public static void main(String[] args) throws IOException {
5.    InputStream input = new FileInputStream("D:\\testout.txt");
6.    DataInputStream inst = new DataInputStream(input);
7.    int count = input.available();
8.    byte[] ary = new byte[count];
9.    inst.read(ary);
10.     for (byte bt : ary) {
11.       char k = (char) bt;
12.       System.out.print(k+"-");
13.     }
14.  }
15.}
```

Here, we are assuming that you have following data in **"testout.txt"** file:
JAVA
Output:
J-A-V-A

Thank You