**Data types**

byte (number, 1 byte)

short (number, 2 bytes)

int (number, 4 bytes)

long (number, 8 bytes)

float (float number, 4 bytes)

double (float number, 8 bytes)

char (a character, 2 bytes)

boolean (true or false, 1 byte)

**Streams**

Java programs perform I/O through streams. A *stream* is an abstraction that either produces

or consumes information. A stream is linked to a physical device by the Java I/O system.

All streams behave in the same manner, even if the actual physical devices to which they are

linked differ. Thus, the same I/O classes and methods can be applied to any type of device.

This means that an input stream can abstract many different kinds of input: from a disk file,

a keyboard, or a network socket. Likewise, an output stream may refer to the console, a disk

file, or a network connection. Streams are a clean way to deal with input/output without

having every part of your code understand the difference between a keyboard and a network,

for example. Java implements streams within class hierarchies defined in the **java.io** package.

**Byte Streams and Character Streams**

Java defines two types of streams: byte and character. Byte streams provide a convenient

means for handling input and output of bytes. Byte streams are used, for example, when

reading or writing binary data. Character streams provide a convenient means for handling

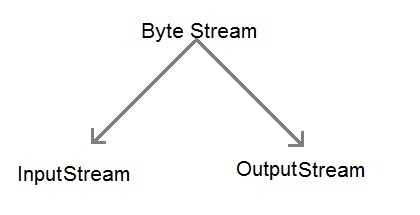
input and output of characters. They use Unicode and, therefore, can be internationalized.

**The Byte Stream Classes**

Byte streams are defined by using two class hierarchies. At the top are two abstract classes:

**InputStream**and **OutputStream**. Each of these abstract classes has several concrete subclasses that handle the differences between various devices, such as disk files, network connections, and even memory buffers

byte: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).

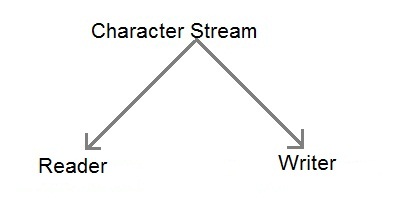


**The Character Stream Classes**

Character streams are defined by using two class hierarchies. At the top are two abstract

classes, **Reader** and **Writer**. These abstract classes handle Unicode character streams. Java

has several concrete subclasses of each of these.



char: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

**INPUTSTREAM AND OUTPUTSTREAM CLASSES**

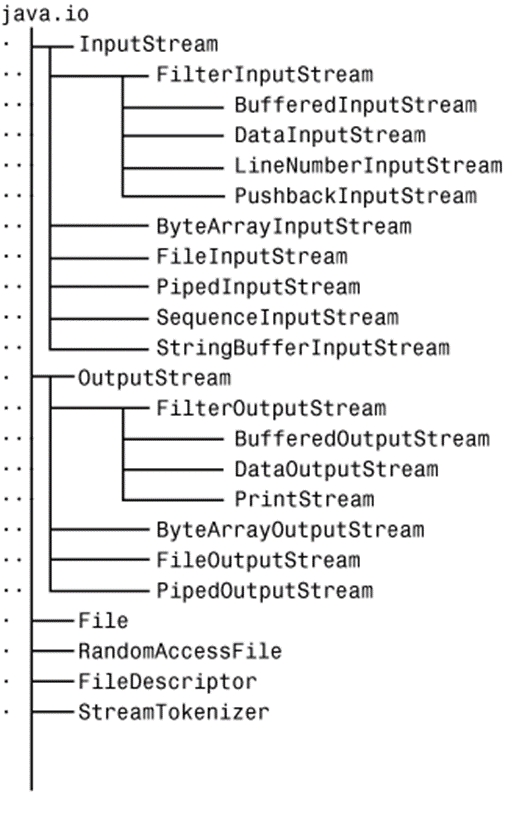
The InputStream and OutputStream classes, and their subclasses are used for dealing with data in binary format. The important methods in two classes are

Important methods in OutputStream class

|  |  |  |
| --- | --- | --- |
| s.no | Syntax of the method | Usage |
| 1 | write() | To write a byte to the OutputStream |
| 2 | write(byte[] b) | To write all bytes in the array **b** to the output stream |
| 3 | write(byte[] b, int n, int m) | To write **m** bytes from array **b** starting from **n**th byte |
| 4 | close() | To close the output stream |
| 5 | flush() | To flush(i.e clear) the output stream |

Important methods in InputStream class

|  |  |  |
| --- | --- | --- |
| s.no | Syntax of the method | Usage |
| 1 | read() | To read a byte from the input stream |
| 2 | read(byte b[]) | To read an array of **b.length**bytes into array **b** |
| 3 | read(byte b[],intn,int m) | To read **m** bytes into array **b** starting from **n**th byte |
| 4 | available() | To give the number of bytes available in the input |
| 5 | skip(n) | To skip over and discard **n** bytes from the input stream |
| 6 | reset() | To go back to the beginning of the stream |
| 7 | close() | To close the input stream |



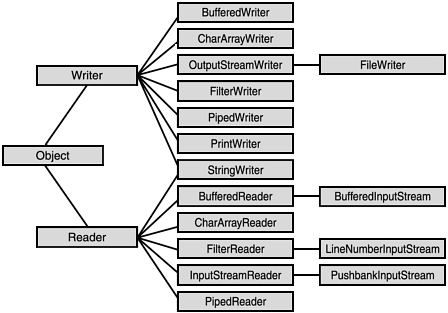


Byte stream class



Character streams

Character stream I/O class



**Reading Console Input**

In Java, console input is accomplished by reading from **System.in**. To obtain a characterbasedstream that is attached to the console, wrap **System.in** in a **BufferedReader**object.

**BufferedReader**supports a buffered input stream

**BufferedReader(Reader *inputReader*)**

Here, *inputReader*is the stream that is linked to the instance of **BufferedReader**that is being

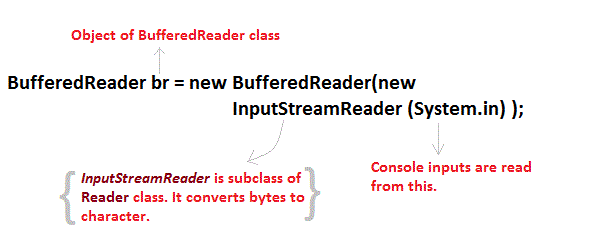
created. **Reader** is an abstract class. One of its concrete subclasses is **InputStreamReader**,

which converts bytes to characters. To obtain an **InputStreamReader**object that is linked to

**System.in**, use the following constructor:

**InputStreamReader(InputStream*inputStream*)**

Because **System.in** refers to an object of type **InputStream**, it can be used for *inputStream.*



After this statement executes, **br**is a character-based stream that is linked to the console

through **System.in**.

this can be written like this also

InputStreamreader reader=new InputStreamReader(system.in);

BufferedReader in=new BufferedReader(reader);

**To read a character from BufferedReader use read()**

The syntax is

intread() throws IOException

example program

|  |
| --- |
| // use of BufferedReader to read characters from console  import java.io.\*;  class BRRead  {  public static void main(String args[]) throws IOException  {  char c;  BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));  System.out.println("Enter characters, 'q' to quit");  do  {  c=(char)br.read();  System.out.println(c);  } while(c!='q');  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>java BRRead  Enter characters, 'q' to quit  123cvbq  1  2  3  c  v  b  q |

Reading Strings

readLine() is used to read strings

syntax

String readLine() throws IOException

|  |
| --- |
| // use of BufferedReader to read characters from console  import java.io.\*;  class BRReadLines  {  public static void main(String args[]) throws IOException  {  String s1;  BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));  System.out.println("Enter lines of text and 'stop' to quit");  do  {  s1=br.readLine();  System.out.println(s1);  } while(!s1.equals("stop"));  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>java BRReadLines  Enter lines of text and 'stop' to quit  welcome  welcome  to  to  java  java  class  class  stop  stop |

To read an array of string values

|  |
| --- |
| // use of BufferedReader to read characters from console  import java.io.\*;  class BRReadLines1  {  public static void main(String args[]) throws IOException  {  String s1[]=new String[100];  BufferedReaderbr=new BufferedReader(new InputStreamReader(System.in));  System.out.println("Enter lines of text and 'stop' to quit");  for(inti=0;i<100;i++)  {  s1[i]=br.readLine();  if(s1[i].equals("stop")) break;  }  System.out.println("the values in array s1 is");  for(inti=0;i<100;i++)  {  if(s1[i].equals("stop")) break;  System.out.println(s1[i]);  }  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac BRReadLines1.java  E:\javaprgs\iostreams>java BRReadLines1  Enter lines of text and 'stop' to quit  this is java class  this is about input output stream  this program is about BufferedReader class  this program reads lines of text as input  stop  the values in array s1 is  this is java class  this is about input output stream  this program is about BufferedReader class  this program reads lines of text as input |

Writing console output

PrintWritter class

Syntax

PrintWriter(OutputStream*outputStream*, Boolean *flushOnNewline*)

The line of code creates a printWriter that is connected to console output

PrintWriter pw=new PrintWriter(System.out, true);

|  |
| --- |
| import java.io.\*;  public class PrintWriterDemo  {  public static void main(String args[])  {  PrintWriter pw=new PrintWriter(System.out, true);  pw.println("This is a string");  inti=-7;  pw.println(i);  double d=4.5e-7;  pw.println(d);  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac PrintWriterDemo.java  E:\javaprgs\iostreams>java PrintWriterDemo  This is a string  -7  4.5E-7 |

Reading and Writing files

Two most often used stream classes for file operation are FileInputStream and FileOutputStream

Syntax

FileInputStream(String *filename*) throws FileNotFoundException

FileOutputStream(String *filename*) throws FileNotFoundException

The *filename* specifies the name of the file that to be opened. If the file does not exists then FileNotfoundException is thrown.

When finished with file, it is to be closed with close() method

Syntax

void close() throws IOException

to read from a file, read() method is used

syntax

intread() throws IOException

FileOutputStream

|  |
| --- |
| //example for FileOutputStream  import java.io.FileOutputStream;  public class filestreamexample1  {  public static void main(String args[])  {  try  {  FileOutputStreamfout=new FileOutputStream("D:\\javaprgs\\iostreams\\testout.txt");  fout.write(65);  String s="welcome to java io stream";  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);  }  } //main()  } |

Output

|  |
| --- |
| D:\javaprgs\iostreams>type testout.txt  Awelcome to java io stream |

FileInputStream

|  |
| --- |
| import java.io.FileInputStream;  public class filestreamexample2  {  public static void main(String args[])  {  try  {  FileInputStream fin=new FileInputStream("D:\\javaprgs\\iostreams\\testout.txt");  inti=0;  while((i=fin.read())!=-1)  {  System.out.print((char)i);  }  fin.close();  }  catch(Exception e)  {  System.out.println(e);  }  }  } |

**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.

|  |
| --- |
| import java.io.\*;  public class Bufferedoutputeg1  {  public static void main(String args[]) throws IOException  {  FileOutputStreamfout=new FileOutputStream("d:\\javaprgs\\iostreams\\buffout.txt");  BufferedOutputStream bout=new BufferedOutputStream(fout);  String s="welcome to iostreams in java";  byte b[]=s.getBytes();  bout.write(b);  bout.flush();  bout.close();  fout.close();  System.out.println("success");  }  } |

Output: The file named buffout.txt is created with the content “welcome to iostreams in java”

BufferedInputStream

|  |
| --- |
| import java.io.\*;  public class Bufferedinputeg1  {  public static void main(String args[]) throws IOException  {  FileInputStream fin=new FileInputStream("d:\\javaprgs\\iostreams\\buffout.txt");  BufferedInputStream bin=new BufferedInputStream(fin);  inti;  while((i=bin.read())!=-1)  {  System.out.print((char)i);  }  bin.close();  fin.close();  }  } |

Output

|  |
| --- |
| D:\javaprgs\iostreams>javac Bufferedinputeg1.java  D:\javaprgs\iostreams>java Bufferedinputeg1  welcome to iostreams in java |

# Java SequenceInputStream Class

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

syntax

|  |  |
| --- | --- |
| 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. |

|  |
| --- |
| import java.io.\*;  class sequenceinputeg1  {  public static void main(String args[]) throws IOException  {  FileInputStream fin1=new FileInputStream("e:\\javaprgs\\iostreams\\testin1.txt");  FileInputStream fin2=new FileInputStream("e:\\javaprgs\\iostreams\\testin2.txt");  SequenceInputStreamseq=new SequenceInputStream(fin1,fin2);  int j;  while((j=seq.read())!=-1)  {  System.out.print((char)j);  }  seq.close();  fin1.close();  fin2.close();  }  } |

Here, 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:

this is an example for sequenceinputstream of io streams in java

After executing the program, you will get following output:

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac sequenceinputeg1.java  E:\javaprgs\iostreams>java -cp . sequenceinputeg1  welcome to java IO programmingthis is an example for sequenceinputstream of io streams in java |

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

In this example, we are writing the data of two files **testin1.txt** and **testin2.txt** into another file named **testout.txt.**

|  |
| --- |
| import java.io.\*;  class sequenceinputeg2  {  public static void main(String args[]) throws IOException  {  FileInputStream fin1=new FileInputStream("e:\\javaprgs\\iostreams\\testin1.txt");  FileInputStream fin2=new FileInputStream("e:\\javaprgs\\iostreams\\testin2.txt");  FileOutputStream fout1=new FileOutputStream("e:\\javaprgs\\iostreams\\testout1.txt");  SequenceInputStreamseq=new SequenceInputStream(fin1,fin2);  int j;  while((j=seq.read())!=-1)  {  fout1.write((char)j);  }  seq.close();  fout1.close();  fin1.close();  fin2.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac sequenceinputeg2.java  E:\javaprgs\iostreams>java -cp . sequenceinputeg2  E:\javaprgs\iostreams>type testout1.txt  welcome to java IO programmingthis is an example for sequenceinputstream of io streams in java |

## **SequenceInputStream example that reads data using enumeration**

If we need to read the data from more than two files, we need to use Enumeration. Enumeration object can be obtained by calling elements() method of the Vector class. Let's see the simple example where we are reading the data from 4 files: a.txt, b.txt, c.txt and d.txt.

|  |
| --- |
| import java.io.\*;  import java.util.\*;  class sequenceinputeg3  {  public static void main(String args[]) throws IOException  {  FileInputStream fin1=new FileInputStream("e:\\javaprgs\\iostreams\\testin1.txt");  FileInputStream fin2=new FileInputStream("e:\\javaprgs\\iostreams\\testin2.txt");  FileInputStream fin3=new FileInputStream("e:\\javaprgs\\iostreams\\testin3.txt");  FileInputStream fin4=new FileInputStream("e:\\javaprgs\\iostreams\\testin4.txt");  FileOutputStream fout1=new FileOutputStream("e:\\javaprgs\\iostreams\\seqout1.txt");  //creating vector object to all the streams  Vector v=new Vector();  v.add(fin1);  v.add(fin2);  v.add(fin3);  v.add(fin4);  //creating enumeration object by calling the elements method  Enumeration e=v.elements();  SequenceInputStreamseq=new SequenceInputStream(e);  int j;  while((j=seq.read())!=-1)  {  fout1.write((char)j);  }  seq.close();  fout1.close();  fin1.close();  fin2.close();  fin3.close();  fin4.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>type seqout1.txt  welcome to java IO programmingthis is an example for sequenceinputstream of io streams in javathis is III computer science classthis is about studying of all iostreams of java |

# 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 declaration**

Let's see the declaration for Java.io.ByteArrayOutputStream class:

1. **public** **class** ByteArrayOutputStream **extends** OutputStream

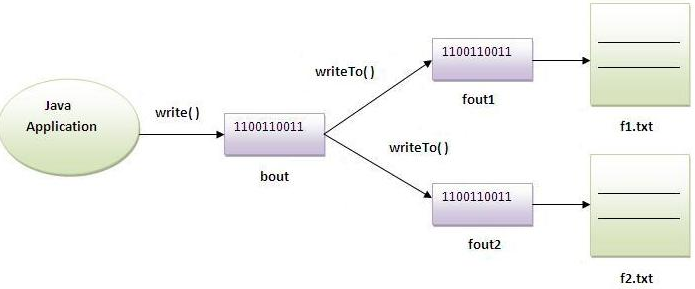
## **Java ByteArrayOutputStream class constructors**

|  |  |
| --- | --- |
| **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. |

|  |
| --- |
| import java.io.\*;  class bytearrayeg1  {  public static void main(String args[]) throws IOException  {  FileOutputStream fout1=new FileOutputStream("e:\\javaprgs\\iostreams\\f1.txt");  FileOutputStream fout2=new FileOutputStream("e:\\javaprgs\\iostreams\\f2.txt");  FileOutputStream fout3=new FileOutputStream("e:\\javaprgs\\iostreams\\f3.txt");  FileOutputStream fout4=new FileOutputStream("e:\\javaprgs\\iostreams\\f4.txt");  ByteArrayOutputStream bout=new ByteArrayOutputStream();  bout.write(65);  bout.writeTo(fout1);  bout.writeTo(fout2);  bout.writeTo(fout3);  bout.writeTo(fout4);  bout.flush();  fout1.close();  fout2.close();  fout3.close();  fout4.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>type f1.txt  A  E:\javaprgs\iostreams>type f2.txt  A  E:\javaprgs\iostreams>type f3.txt  A  E:\javaprgs\iostreams>type f4.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 declaration**

Let's see the declaration for Java.io.ByteArrayInputStream class:

1. **public** **class** ByteArrayInputStream **extends** InputStream

## **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, intlen) | 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. |

|  |
| --- |
| import java.io.\*;  class bytearrayeg2  {  public static void main(String args[]) throws IOException  {  byte b[]={35,36,37,38,39,40};  ByteArrayInputStream bin=new ByteArrayInputStream(b);  int k=0;  while((k=bin.read())!=-1)  {  char ch=(char)k; // conversion of a byte into character  System.out.println("ASCII value of character : "+k+"is "+ch);  }  }  } |

output

|  |
| --- |
| E:\javaprgs\iostreams>javac bytearrayeg2.java  E:\javaprgs\iostreams>java -cp . bytearrayeg2  ASCII value of character : 35is #  ASCII value of character : 36is $  ASCII value of character : 37is %  ASCII value of character : 38is &  ASCII value of character : 39is '  ASCII value of character : 40is ( |

# 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 declaration**

Let's see the declaration for java.io.DataOutputStream class:

1. **public** **class** DataOutputStream **extends** FilterOutputStream **implements** DataOutput

## **Java DataOutputStream class methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| intsize() | 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, intlen) | 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. |

|  |
| --- |
| import java.io.\*;  class datastreameg1  {  public static void main(String args[]) throws IOException  {  FileOutputStreamfout=new FileOutputStream("e:\\javaprgs\\iostreams\\dataout.txt");  DataOutputStream data=new DataOutputStream(fout);  data.writeInt(65);  boolean b=true;  data.writeBoolean(b);  data.writeChar('a');  data.writeChars("welcome");  data.flush();  data.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>java -cp . datastreameg1  E:\javaprgs\iostreams>type dataout.txt  Aa w e l c o m e |

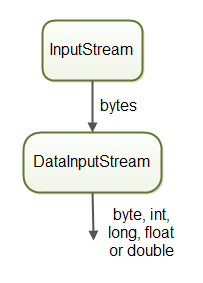
# Java DataInputStream Class

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.

## **Java DataInputStream class declaration**

Let's see the declaration for java.io.DataInputStream class:

**public** **class** DataInputStream **extends** FilterInputStream **implements** DataInput

**DataOutputStream and DataInputStream classes**

The DataInputStream class enables to read Java primitives from InputStream's instead of only bytes. You wrap an InputStream in a DataInputStream and then you can read primitives from it. DataOutputStream and DataInputStream give us the power to write and read primitive data type to a media such as file. Both of this class have the corresponding method to write primitive data and read it back.

Using this class make it easier to read integer, float, double data and others without needing to interpret if the read data should be an integer or a float data..

Create an instance of FileOutputStream with cities.dat as the file name to be created. Then we pass the input stream object in the DataOutputStream constructor. Below we write some data to the cities.dat. DataOutputStream class have various method that allow us to write primitive type data and string. There are method called writeInt(), writeFloat(), writeUTF(), etc.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | import java.io.\*;  public class datainputstreamdemo  {  public static void main(String[] args)  {  int stateCode1 = 44;  String stateName1 = "Tamil Nadu";  int statePopulation1 = 72138958;  float stateTemp1 = 40f;  try {  FileOutputStream fos = new FileOutputStream("cities.dat");  DataOutputStream dos = new DataOutputStream(fos);  dos.writeInt(stateCode1);  dos.writeUTF(stateName1);  dos.writeInt(statePopulation1);  dos.writeFloat(stateTemp1);    dos.flush();  dos.close();  FileInputStream fis = new FileInputStream("cities.dat");  DataInputStream dis = new DataInputStream(fis);  int stateCode = dis.readInt();  System.out.println("Id: " + stateCode);  String stateName = dis.readUTF();  System.out.println("Name: " + stateName);  int statePopulation = dis.readInt();  System.out.println("Population: " + statePopulation);  float stateTemp = dis.readFloat();  System.out.println("Temperature: " + stateTemp);    } catch (IOException e) {  e.printStackTrace();  }  }  } |

**Output**

|  |
| --- |
| **Id: 44**  **Name: Tamil Nadu**  **Population: 72138958**  **Temperature: 40.0** |

## **Java DataInputStream class Methods**

|  |  |
| --- | --- |
| **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, intlen) | It is used to read **len** bytes of data from the input stream. |
| intreadInt() | 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. |
| booleanreadBoolean() | It is used to read one input byte and return true if byte is non zero, false if byte is zero. |
| intskipBytes(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, intlen) | It is used to read **len** bytes from the input stream. |

# Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

Let's see a simple example to read text from console.

1. String text=System.console().readLine();
2. System.out.println("Text is: "+text);

## **Java Console class declaration**

Let's see the declaration for Java.io.Console class:

1. **public** **final** **class** Console **extends** Object **implements** Flushable

## **Java Console class methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| Reader reader() | It is used to retrieve the reader object associated with the console |
| String readLine() | It is used to read a single line of text from the console. |
| String readLine(String fmt, Object... args) | It provides a formatted prompt then reads the single line of text from the console. |
| char[] readPassword() | It is used to read password that is not being displayed on the console. |
| char[] readPassword(String fmt, Object... args) | It provides a formatted prompt then reads the password that is not being displayed on the console. |
| Console format(String fmt, Object... args) | It is used to write a formatted string to the console output stream. |
| Console printf(String format, Object... args) | It is used to write a string to the console output stream. |
| PrintWriterwriter() | It is used to retrieve the PrintWriter object associated with the console. |
| void flush() | It is used to flushes the console. |

|  |
| --- |
| import java.io.Console;  class consoleinput1  {  public static void main(String args[])  {  Console c=System.console();  System.out.println("enter your name");  String n=c.readLine();  System.out.println("welcome"+n);  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>java -cp . consoleinput1  enter your name  radha  welcomeradha |

## **Java Console Example to read password**

|  |
| --- |
| import java.io.Console;  class consoleinput2  {  public static void main(String args[])  {  Console c=System.console();  System.out.println("enter your name");  String n=c.readLine();  System.out.println("enter Password");  char[] ch=c.readPassword();  String pass=String.valueOf(ch);  System.out.println("welcome"+n);  System.out.println("password is : "+pass);  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac consoleinput2.java  E:\javaprgs\iostreams>java -cp . consoleinput2  enter your name  radha  enter Password  welcomeradha  password is :liril |

READER and WRITER classes

The Reader and Writer classes, and their subclasses are used for dealing with data in text format. While

Java Writer

It is an abstract class for writing to character streams.

### Methods

|  |  |  |
| --- | --- | --- |
| 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(char[] cbuf) | It writes an array of characters. |
| abstract void | write(char[] cbuf, int off, int len) | It writes a portion of an array of characters. |
| void | write(int c) | It writes a single character. |
| void | write(String str) | It writes a string. |
| void | write(String str, int off, int len) | It writes a portion of a string. |

## **Java Writer Example**

|  |
| --- |
| import java.io.\*;public class writereg1{public static void main(String args[]) throws IOException{Writer w = new FileWriter("e:\\javaprgs\\iostreams\\writer1.text");String str="I love my country";w.write(str);w.close();}} |

Output

|  |
| --- |
| E:\javaprgs\iostreams>type writer1.text  I love my country |

# Java Reader

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close().

### Constructor

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Constructor** | **Description** |
| protected | Reader() | It creates a new character-stream reader whose critical sections will synchronize on the reader itself. |
| protected | Reader(Object lock) | It creates a new character-stream reader whose critical sections will synchronize on the given object. |

### Methods

|  |  |  |
| --- | --- | --- |
| **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. |
| boolean | markSupported() | It tells whether this stream supports the mark() operation. |
| int | read() | It reads a single character. |
| int | read(char[] cbuf) | It reads characters into an array. |
| abstract int | read(char[] cbuf, int off, int len) | It reads characters into a portion of an array. |
| int | read(CharBuffer target) | It attempts to read characters into the specified character buffer. |
| 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. |

Reader example

|  |
| --- |
| import java.io.\*;  public class readereg1  {  public static void main(String args[]) throws IOException  {  Reader r = new FileReader("e:\\javaprgs\\iostreams\\writer1.text");  int data=r.read();  while(data!=-1)  {  System.out.print((char)data);  data=r.read();  }  r.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac readereg1.java  E:\javaprgs\iostreams>java -cp . readereg1  I love my country |

# Java 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 of FileWriter class**

|  |  |
| --- | --- |
| **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. |

## **Methods of FileWriter class**

|  |  |
| --- | --- |
| **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 example

|  |
| --- |
| import java.io.\*;  public class filewritereg1  {  public static void main(String args[]) throws IOException  {  FileWriter fw = new FileWriter("e:\\javaprgs\\iostreams\\writer2.txt");  fw.write("welcome to filewriter class in java");  fw.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>type writer2.txt  welcome to filewriter class in java |

# Java 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 of FileReader class**

|  |  |
| --- | --- |
| **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. |

## **Methods of FileReader class**

|  |  |
| --- | --- |
| **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. |

## **Java FileReader Example**

|  |
| --- |
| import java.io.\*;  public class filereadereg1  {  public static void main(String args[]) throws IOException  {  FileReader fr = new FileReader("e:\\javaprgs\\iostreams\\writer2.txt");  int i;  while((i=fr.read())!=-1)  {  System.out.print((char)i);  }  fr.close();  }  } |

Output

|  |
| --- |
| E:\javaprgs\iostreams>javac filereadereg1.java  E:\javaprgs\iostreams>java -cp . filereadereg1  welcome to filewriter class in java |

# Java BufferedWriter Class

Java BufferedWriter class 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.

## **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. |

## **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 |

**Example**

|  |
| --- |
| **import java.io.\*;**  **public class bufferedwritereg1**  **{**  **public static void main(String args[]) throws IOException**  **{**  **FileWriter fw = new FileWriter("e:\\javaprgs\\iostreams\\writer3.txt");**  **BufferedWriter buff=new BufferedWriter(fw);**  **buff.write("i love java");**  **buff.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>java -cp . bufferedwritereg1**  **E:\javaprgs\iostreams>type writer3.txt**  **i love java** |

# Java BufferedReader Class

Java BufferedReader class 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.

## **Java BufferedReader class**

|  |  |
| --- | --- |
| **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. |

example

|  |
| --- |
| **import java.io.\*;**  **public class bufferedreadereg1**  **{**  **public static void main(String args[]) throws IOException**  **{**  **FileReader fr = new FileReader("e:\\javaprgs\\iostreams\\writer3.txt");**  **BufferedReader buff=new BufferedReader(fr);**  **int i;**  **while((i=buff.read())!=-1)**  **{**  **System.out.print((char)i);**  **}**  **buff.close();**  **fr.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac bufferedreadereg1.java**  **E:\javaprgs\iostreams>java -cp . bufferedreadereg1**  **i love java** |

## **Reading data from console by InputStreamReader and BufferedReader**

In this example, we are connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

**Example**

|  |
| --- |
| **import java.io.\*;**  **public class bufferedreadereg2**  **{**  **public static void main(String args[]) throws IOException**  **{**  **InputStreamReader fr = new InputStreamReader(System.in);**  **BufferedReader buff=new BufferedReader(fr);**  **System.out.println("enter your name");**  **String name=buff.readLine();**  **System.out.println("welcome "+name);**  **buff.close();**  **fr.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac bufferedreadereg2.java**  **E:\javaprgs\iostreams>java -cp . bufferedreadereg2**  **enter your name**  **radha**  **welcome radha** |

# Java CharArrayReader Class

The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character array as a reader (stream). It inherits Reader class.

## **ava CharArrayReader class methods**

|  |  |
| --- | --- |
| **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. |

**Example**

|  |
| --- |
| **import java.io.\*;**  **public class chararrayreadeg2**  **{**  **public static void main(String args[]) throws IOException**  **{**  **char[] a1={'w','e','l','c','o','m','e'};**  **CharArrayReader cr=new CharArrayReader(a1);**  **int k=0;**  **while((k=cr.read())!=-1)**  **{**  **char ch=(char)k;**  **System.out.println(ch+":"+k);**  **}**  **cr.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac chararrayreadeg2.java**  **E:\javaprgs\iostreams>java -cp . chararrayreadeg2**  **w:119**  **e:101**  **l:108**  **c:99**  **o:111**  **m:109**  **e:101** |

# Java CharArrayWriter Class

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.

## **Java CharArrayWriter class Methods**

|  |  |
| --- | --- |
| **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. |

**Example**

|  |
| --- |
| **import java.io.\*;**  **public class chararraywriteeg1**  **{**  **public static void main(String args[]) throws IOException**  **{**  **CharArrayWriter cw=new CharArrayWriter();**  **cw.write("welcome to chararraywriter demo");**  **FileWriter f1=new FileWriter("e:\\javaprgs\\iostreams\\a.txt");**  **FileWriter f2=new FileWriter("e:\\javaprgs\\iostreams\\b.txt");**  **FileWriter f3=new FileWriter("e:\\javaprgs\\iostreams\\c.txt");**  **FileWriter f4=new FileWriter("e:\\javaprgs\\iostreams\\d.txt");**  **cw.writeTo(f1);**  **cw.writeTo(f2);**  **cw.writeTo(f3);**  **cw.writeTo(f4);**  **cw.close();**  **f1.close();**  **f2.close();**  **f3.close();**  **f4.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac chararraywriteeg1.java**  **E:\javaprgs\iostreams>java -cp . chararraywriteeg1**  **E:\javaprgs\iostreams>type a.txt**  **welcome to chararraywriter demo**  **E:\javaprgs\iostreams>type b.txt**  **welcome to chararraywriter demo**  **E:\javaprgs\iostreams>type c.txt**  **welcome to chararraywriter demo**  **E:\javaprgs\iostreams>type d.txt**  **welcome to chararraywriter demo** |

# Java PrintStream Class

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

## **Methods of PrintStream class**

|  |  |
| --- | --- |
| **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. |
| void println() | It terminates the line only. |
| void printf(Object format, Object... args) | It writes the formatted string to the current stream. |
| void printf(Locale l, Object format, Object... args) | It writes the formatted string to the current stream. |
| void format(Object format, Object... args) | It writes the formatted string to the current stream using specified format. |
| void format(Locale l, Object format, Object... args) | It writes the formatted string to the current stream using specified format. |

**Example**

|  |
| --- |
| **import java.io.\*;**  **public class printstream1**  **{**  **public static void main(String args[]) throws IOException**  **{**  **FileOutputStream fout=new FileOutputStream("e:\\javaprgs\\iostreams\\printstream.txt");**  **PrintStream pout=new PrintStream(fout);**  **int a=19;**  **double pi=Math.PI;**  **pout.println(2018);**  **pout.println("welcome");**  **pout.println("to java");**  **System.out.printf("%d%n",a);**  **System.out.printf("%f %n",pi);**  **System.out.printf("%.3f %n",pi);**  **System.out.printf("%10.3f %n",pi);**  **pout.printf("%d%n",a);**  **pout.printf("%f %n",pi);**  **pout.printf("%.3f %n",pi);**  **pout.printf("%10.3f %n",pi);**  **pout.close();**  **fout.close();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac printstream1.java**  **E:\javaprgs\iostreams>java -cp . printstream1**  **19**  **3.141593**  **3.142**  **3.142**  **E:\javaprgs\iostreams>type printstream.txt**  **2018**  **welcome**  **to java**  **19**  **3.141593**  **3.142**  **3.142** |

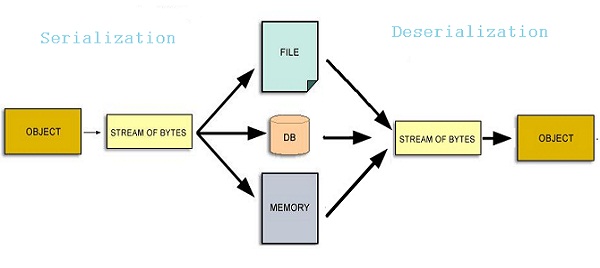
**Serialization**

### Serialization and Deserialization in Java

**Serialization** is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams. The reverse process of creating object from sequence of bytes is called **deserialization**.

A class must implement **Serializable** interface present in java.io package in order to serialize its object successfully. **Serializable** is a **marker interface** that adds serializable behaviour to the class implementing it.

Only the objects of those classes can be serialized which are implementing **java.io.Serializable** interface.  
Serializable is a **marker interface** (has no data member and method). It is used to “mark” java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are:- Cloneable and Remote.

****

#### Serializing an Object

|  |
| --- |
| import java.io.\*;  class studentinfo implements Serializable  {  String name;  int rollno;  String mobile;  studentinfo(String n, int r,String m)  {  this.name=n;  this.rollno=r;  this.mobile=m;  }  }  class serializable1  {  public static void main(String args[]) throws IOException  {  studentinfo si=new studentinfo("anitha",100,"9988776655");  FileOutputStream fos=new FileOutputStream("e:\\javaprgs\\iostreams\\student.ser");  ObjectOutputStream oos=new ObjectOutputStream(fos);  oos.writeObject(si);  oos.close();  fos.close();  }  } |

#### Deserialization of Object

|  |
| --- |
| **import java.io.\*;**  **class serializable2**  **{**  **public static void main(String args[])**  **{**  **studentinfo si=null;**  **try**  **{**  **FileInputStream fis=new FileInputStream("e:\\javaprgs\\iostreams\\student.ser");**  **ObjectInputStream ois=new ObjectInputStream(fis);**  **si=(studentinfo)ois.readObject();**  **}**  **catch(Exception e)**  **{**  **System.out.println(e);**  **}**  **System.out.println("name :"+si.name);**  **System.out.println("rollno :"+si.rollno);**  **System.out.println("mobile :"+si.mobile);**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs\iostreams>javac -cp . serializable2.java**  **E:\javaprgs\iostreams>java -cp . serializable2**  **name :anitha**  **rollno :100**  **mobile :9988776655** |

Object streams

The object streams enable us to store the data as object and retrieve as object. The process of saving an object to a stream is called serialization. This is because of the fact that each object is assigned a serial number on the stream. When an object is read back from the disk, a duplicate serial number is restored as reference to the same object.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | import java.io.\*;  class student implements Serializable  {  String stname;  int rollno;  int mark1,mark2;  int total;  float avg;  student(String sn,int r,int m1,int m2)  {  stname=sn;  rollno=r;  mark1=m1;  mark2=m2;  total=mark1+mark2;  avg=total/2;  }  public String toString()  {  String text="student Name= "+stname+ "\nroll no : "+rollno;  text=text+"\nmark1 : "+mark1+"\nmark2 : "+mark2+"\ntotal "+total+"\navg "+avg;  return text;  }  }  public class studentobject  {  public static void main(String args[])  {  // object serialization  try  {  student obj1=new student("xxx",8851,75,80);  FileOutputStream fos=new FileOutputStream("serial1");  ObjectOutputStream oos= new ObjectOutputStream(fos);  oos.writeObject(obj1);  oos.flush();  oos.close();  }  catch(Exception e)  {  System.out.println("Exception :"+e);  System.exit(0);  }  // object deserialization  try  {  student obj2;  FileInputStream fis=new FileInputStream("serial1");  ObjectInputStream ois= new ObjectInputStream(fis);  obj2=(student)ois.readObject();  ois.close();  System.out.println("object :"+obj2);  }  catch(Exception e)  {  System.out.println("Exception :"+e);  System.exit(0);  }  }  } |

Output

|  |
| --- |
| object :student Name= xxx  roll no : 8851  mark1 : 75  mark2 : 80  total 155  avg 77.0 |

When more than one object to be written to a file see the example below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | import java.io.\*;  class student implements Serializable  {  String stname;  int rollno;  int mark1,mark2;  int total;  float avg;  student(String sn,int r,int m1,int m2)  {  stname=sn;  rollno=r;  mark1=m1;  mark2=m2;  total=mark1+mark2;  avg=total/2;  }  public String toString()  {  String text="student Name= "+stname+ "\nroll no : "+rollno;  text=text+"\nmark1 : "+mark1+"\nmark2 : "+mark2+"\ntotal "+total+"\navg "+avg;  return text;  }  }  public class studentobject2  {  public static void main(String args[])  {  InputStreamReader inp = new InputStreamReader(System.in);  BufferedReader br = new BufferedReader(inp);  // object serialization  try  {  FileOutputStream fos=new FileOutputStream("serial1");  ObjectOutputStream oos= new ObjectOutputStream(fos);  for(int i=0;i<2;i++)  {  System.out.println("enter name:");  String str = br.readLine();  System.out.println("rollno:");  int r = Integer.parseInt(br.readLine());  System.out.println("mark1:");  int m1 = Integer.parseInt(br.readLine());  System.out.println("mark2:");  int m2 = Integer.parseInt(br.readLine());  student obj1=new student(str,r,m1,m2);  oos.writeObject(obj1);  }  oos.flush();  oos.close();  }  catch(Exception e)  {  System.out.println("Exception :"+e);  System.exit(0);  }  // object deserialization  try  {  student obj2;  FileInputStream fis=new FileInputStream("serial1");  ObjectInputStream ois= new ObjectInputStream(fis);  for(int i=0;i<2;i++)  {  obj2=(student)ois.readObject();  System.out.println("object :"+obj2);  }  ois.close();  }  catch(Exception e)  {  System.out.println("Exception :"+e);  System.exit(0);  }  }  } |

Output

|  |
| --- |
| enter name:  xx  rollno:  123  mark1:  56  mark2:  90  enter name:  yy  rollno:  124  mark1:  89  mark2:  78  object :student Name= xx  roll no : 123  mark1 : 56  mark2 : 90  total 146  avg 73.0  object :student Name= yy  roll no : 124  mark1 : 89  mark2 : 78  total 167  avg 83.0  Process completed. |