File Handling I

✓ In Java, a File is an abstract data type. A named location used to store related information is known as a File.

✓ There are several File Operations like creating a new File, getting information about File, writing into a File, reading from a File and deleting a File.

✓ Before understanding the File operations, it is required that we should have knowledge of Stream and File methods.

File Handling II

S.No.	Method	Return Type	Description
1.	canRead()	Boolean	The canRead() method is used to check whether we can read the data of the file or not.
2.	createNewFile()	Boolean	The createNewFile() method is used to create a new empty file.
3.	canWrite()	Boolean	The canWrite() method is used to check whether we can write the data into the file or not.
4.	exists()	Boolean	The exists() method is used to check whether the specified file is present or not.
5.	delete()	Boolean	The delete() method is used to delete a file.
6.	getName()	String	The getName() method is used to find the file name.
7.	getAbsolutePath()	String	The getAbsolutePath() method is used to get the absolute pathname of the file.
8.	length()	Long	The length() method is used to get the size of the file in bytes.
9.	list()	String[]	The list() method is used to get an array of the files available in the directory.
10.	mkdir()	Boolean	The mkdir() method is used for creating a new directory.

File Operations

- Create a File
- Get File Information
- Write to a File
- Read from a File
- Delete a File

File Handling IV

Create File: Create a File operation is performed to create a new file. We use the createNewFile() method of file. The createNewFile() method returns true when it successfully creates a new file and returns false when the file already exists.

File Handling V

```java
// Importing File class
import java.io.File;
// Importing the IOException class for handling errors
import java.io.IOException;
class CreateFile
{
    public static void main(String args[])
    {
        try
        {
            // Creating an object of a file
            File f0 = new File("D:FileOperationExample.txt");
            if (f0.createNewFile())
            {
                System.out.println("File " + f0.getName() + " is created successfully.");
            }
            else
            {
                System.out.println("File is already exist in the directory.");
            }
        }
        catch (IOException exception)
        {
            System.out.println("An unexpected error is occurred.");
            exception.printStackTrace();
        }
    }
}
```

File Handling VI

Get File Information: The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

File Handling VII

```
1// Import the File class
2import java.io.File;
3class FileInfo
4{
5       public static void main(String[] args)
6       {
7               // Creating file object
8               File f0 = new File("D:FileOperationExample.txt");
9               if (f0.exists())
10              {
11                      // Getting file name
12                      System.out.println("The name of the file is:"
        + f0.getName());
13
14                      // Getting path of the file
15                      System.out.println("The absolute path of the
        file is:" + f0.getAbsolutePath());
16
17                      // Checking whether the file is writable or not
18                      System.out.println("Is file writeable?: " + f0.
        canWrite());
19
20                      // Checking whether the file is readable or not
21                      System.out.println("Is file readable " + f0.
        canRead());
22
23                      // Getting the length of the file in bytes
24                      System.out.println("The size of the file in
        bytes is:" + f0.length());
25              }
26              else
27              {
28                      System.out.println("The file does not exist.");
29              }
30      }
31}
```

File Handling VIII

Write to a File: The next operation which we can perform on a file is "writing into a file". In order to write data into a file, we will use the FileWriter class and its write() method together. We need to close the stream using the close() method to retrieve the allocated resources.

File Handling IX

```
1// Importing the FileWriter class
2import java.io.FileWriter;
3
4// Importing the IOException class for handling errors
5import java.io.IOException;
6
7class WriteToFile
8{
9       public static void main(String[] args)
10      {
11
12              try
13              {
14                      FileWriter fwrite = new FileWriter("D:
        FileOperationExample.txt");
15                      // writing the content into the
        FileOperationExample.txt file
16                      fwrite.write("A named location used to store
    related information is referred to as a File.");
17
18                      // Closing the stream
19                      fwrite.close();
20                      System.out.println("Content is successfully
    wrote to the file.");
21              }
22              catch (IOException e)
23              {
24                      System.out.println("Unexpected error occurred")
    ;
25                      e.printStackTrace();
26              }
27      }
28}
```

File Handling X

Read from a File: The next operation which we can perform on a file is "read from a file". In order to write data into a file, we will use the Scanner class. Here, we need to close the stream using the close() method. We will create an instance of the Scanner class and use the hasNextLine() method nextLine() method to get data from the file.

File Handling XI

```
1// Importing the File class
2import java.io.File;
3// Importing FileNotFoundException class for handling errors
4import java.io.FileNotFoundException;
5// Importing the Scanner class for reading text files
6import java.util.Scanner;
7
8class ReadFromFile
9{
10      public static void main(String[] args)
11      {
12              try
13              {
14                      // Create f1 object of the file to read data
15                      File f1 = new File("D:FileOperationExample.txt")
        ;
16                      Scanner dataReader = new Scanner(f1);
17                      while (dataReader.hasNextLine())
18                      {
19                              String fileData = dataReader.nextLine();
20                              System.out.println(fileData);
21                      }
22                      dataReader.close();
23              }
24              catch (FileNotFoundException exception)
25              {
26                      System.out.println("Unexcpected␣error␣occurred!
    ");
27                      exception.printStackTrace();
28              }
29      }
30}
```

File Handling XII

Delete a File: The next operation which we can perform on a file is "deleting a file".

- ✓ In order to delete a file, we will use the delete() method of the file.
- ✓ We don't need to close the stream using the close() method because for deleting a file, we neither use the FileWriter class nor the Scanner class.

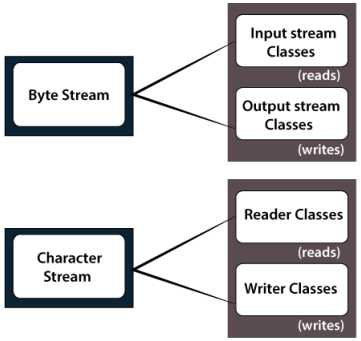
```
1 // Importing the File class
2 import java.io.File;
3 class DeleteFile
4 {
5     public static void main(String[] args)
6     {
7         File f0 = new File("D:
          FileOperationExample.txt");
8         if (f0.delete())
9         {
10            System.out.println(f0.
              getName()+ " file is deleted
              successfully.");
11        }
12        else
13        {
14            System.out.println("
              Unexpected error found in deletion
              of the file.");
15        }
16    }
17 }
```

File Handling XIII

I/O stream I

Stream

✓ A series of data is referred to as a stream. In Java, Stream is classified into two types, i.e., BYTE STREAM AND CHARACTER STREAM.



Brief classification of I/O streams

I/O stream II

Aspect	Byte Stream	Character Stream
Data Type	Binary data (8-bit bytes).	Text data (16-bit characters).
Classes	InputStream and OutputStream.	Reader and Writer.
Use Case	Non-text files (images, videos).	Text files (plain text, JSON, XML).
Encoding Support	No encoding/decoding. Direct bytes. vHandles character encoding (e.g., UTF-8).	
Performance	Faster for binary data.	Better for text-based data.

I/O stream III

✓ In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1. System.out: standard output stream
2. System.in: standard input stream
3. System.err: standard error stream

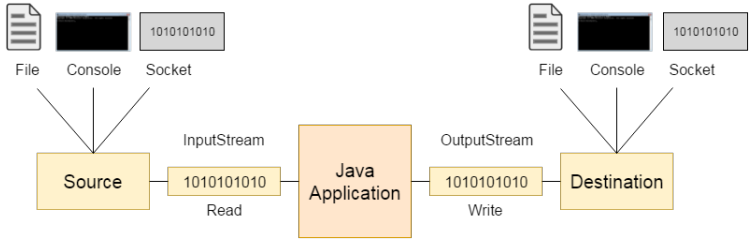
```
System.out.println("simple message");
System.err.println("error message");
```

```
int i=System.in.read(); //returns ASCII code of 1st character
System.out.println((char)i); //will print the character
```

OutputStream vs InputStream

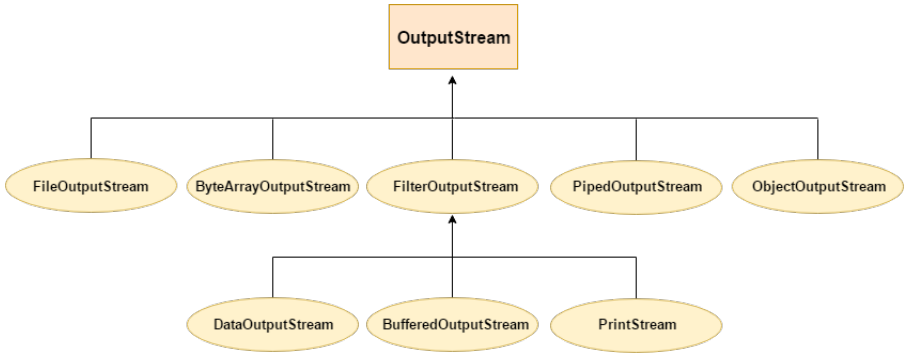
✓ Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

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



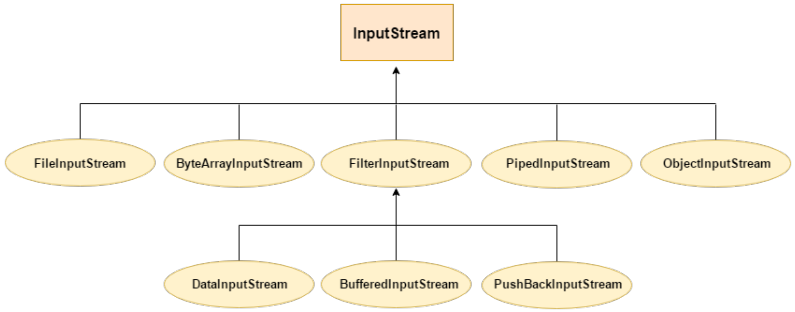
I/O stream V

OutputStream Hierarchy: 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.



I/O stream VI

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



Basic I/O Stream

1. Byte Streams: handle I/O of raw binary data.
2. Character Streams: handle I/O of character data, automatically handling translation to and from the local character set.
3. Buffered Streams: optimize input and output by reducing the number of calls to the native API.
4. Scanning and Formatting: allows a program to read and write formatted text.
5. I/O from the Command Line: describes the Standard Streams and the Console object.
6. Data Streams: handle binary I/O of primitive data type and String values.
7. Object Streams: handle binary I/O of objects.

Byte Streams

Programs use byte streams to perform input and output of 8-bit bytes. All byte stream classes are descended from InputStream and OutputStream.

✓ There are many byte stream classes.

✓ The file I/O byte streams, FileInputStream and FileOutputStream. Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

InputStream	OutputStream
FIleInputStream	FileOutputStream
ByteArrayInputStream	ByteArrayOutputStream
ObjectInputStream	ObjectOutputStream
PipedInputStream	PipedOutputStream
FilteredInputStream	FilteredOutputStream
BufferedInputStream	BufferedOutputStream
DataInputStream	DataOutputStream

FileInputStream and FileOutputStream:

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

Table 1: methods of InputStream

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

Table 2: methods of OutputStream

I/O stream XI

✓ FileInputStream and FileOutputStream which use byte streams to copy Input.txt, one byte at a time.

```
1import java.io.FileInputStream;
2import java.io.FileOutputStream;
3import java.io.IOException;
4public class CopyBytes
5{
6       public static void main(String[] args) throws IOException
7       {
8               FileInputStream in = null;
9               FileOutputStream out = null;
10              try
11              {
12                      in = new FileInputStream("Input.txt");
13                      out = new FileOutputStream("output.txt");
14                      int c;
15                      while ((c = in.read()) != —1)
16                      {
17                              out.write(c);
18                      }
19              }
20              finally
21              {
22                      if (in != null)
23                      {
24                              in.close();
25                      }
26                      if (out != null)
27                      {
28                              out.close();
29                      }
30              }
31      }
32}
```

I/O stream XII

ByteArrayInputStream: 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.

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.

Table 3: Constructor of ByteArrayInputStream

I/O stream XIII

Method	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.

Table 4: Methods of ByteArrayInputStream

I/O stream XIV

```
1//ByteArrayInputStream class to read byte array as input stream.
2import java.io.*;
3public class ReadExample
4{
5       public static void main(String[] args) throws IOException
6       {
7               byte[] buf =
8               {
9                       35, 36, 37, 38
10              }
11              ;
12              // Create the new byte array input stream
13              ByteArrayInputStream byt = new ByteArrayInputStream(buf);
14              int k = 0;
15              while ((k = byt.read()) != -1)
16              {
17                      //Conversion of a byte into character
18                      char ch = (char) k;
19                      System.out.println("ASCII value of Character is:" + k + "; Special character is:" + ch);
20              }
21      }
22}
23//Output:
24ASCII value of Character is:35; Special character is: #
25ASCII value of Character is:36; Special character is: $
26ASCII value of Character is:37; Special character is: %
27ASCII value of Character is:38; Special character is: &
```

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.

I/O stream XVI

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.

Table 5: Constructor of ByteArrayOutputStream

I/O stream XVII

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.

Table 6: Methods of ByteArrayOutputStream

I/O stream XVIII

```
1import java.io.*;
2public class DataStreamExample
3{
4    public static void main(String args[])throws Exception
5    {
6        FileOutputStream fout1=new FileOutputStream("D:\\f1.txt");
7        FileOutputStream fout2=new FileOutputStream("D:\\f2.txt");
8
9        ByteArrayOutputStream bout=new ByteArrayOutputStream();
10       bout.write(65);
11       bout.writeTo(fout1);
12       bout.writeTo(fout2);
13
14       bout.flush();
15       bout.close();
16       System.out.println("Success...");
17   }
18}
```

Always Close Streams

✓ Closing a stream when it's no longer needed is very important — so important that CopyBytes uses a finally block to guarantee that both streams will be closed even if an error occurs. This practice helps avoid serious resource leaks.

✓ One possible error is that CopyBytes was unable to open one or both files. When that happens, the stream variable corresponding to the file never changes from its initial null value. That's why CopyBytes makes sure that each stream variable contains an object reference before invoking close.

When Not to Use Byte Streams

✓ CopyBytes seems like a normal program, but it actually represents a kind of low-level I/O that you should avoid.

✓ There are also streams for more complicated data types. Byte streams should only be used for the most primitive I/O.

Another example:

```
1import java.io.File;
2import java.io.FileInputStream;
3import java.io.FileOutputStream;
4import java.io.IOException;
5public class IOStreamsExample
6{
7     public static void main(String args[]) throws IOException
8     {
9          //Creating FileInputStream object
10         File file = new File("D:/myFile.txt");
11         FileInputStream fis = new FileInputStream(file);
12         byte bytes[] = new byte[(int) file.length()];
13         //Reading data from the file
14         fis.read(bytes);
```

I/O stream XXI

```
15          //Writing data to another file
16          File out = new File("D:/CopyOfmyFile.txt");
17          FileOutputStream outputStream = new FileOutputStream(out);
18          //Writing data to the file
19          outputStream.write(bytes);
20          outputStream.flush();
21          System.out.println("Data␣successfully␣written␣in␣the␣specified␣file");
22      }
23  }
```

I/O stream XXII

Character Streams: These handle data in 16 bit Unicode. Using these you can read and write text data only.

✓ The Reader and Writer classes (abstract) are the super classes of all the character stream classes: classes that are used to read/write character streams. Following are the character array stream classes provided by Java.

Reader	Writer
BufferedReader	BufferedWriter
CharacterArrayReader	CharacterArrayWriter
StringReader	StringWriter
FileReader	FileWriter
InputStreamReader	InputStreamWriter

I/O stream XXIII

BufferedReader: 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.

Constructor name	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.

Table 7: Constructor

I/O stream XXIV

Method name	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.

I/O stream XXV

```
1import java.io.*;
2public class BufferedReaderExample
3{
4    public static void main(String args[])throws Exception
5    {
6        FileReader fr=new FileReader("D:\\testout.txt");
7        BufferedReader br=new BufferedReader(fr); //It is used to create a buffered character input stream
           that uses the default size for an input buffer.
8        int i;
9        while((i=br.read())!=-1)
10       {
11           System.out.print((char)i);
12       }
13       br.close();
14       fr.close();
15   }
16}
```

I/O stream XXVI

```
1 import java.io.*;
2
3 public class MarkSupportedExample
4 {
5     public static void main(String[] args)
6     {
7         try
8         {
9             // Using ByteArrayInputStream (supports mark/reset)
10            byte[] data = "Hello World".getBytes();
11            InputStream input1 = new ByteArrayInputStream(data);
12
13            System.out.println("ByteArrayInputStream supports mark: " + input1.markSupported()); //
                  true
14
15            // Using FileInputStream (doesn't support mark/reset)
16            InputStream input2 = new FileInputStream("example.txt");
17
18            System.out.println("FileInputStream supports mark: " + input2.markSupported()); // false
19
```

I/O stream XXVII

```
20              // Using BufferedInputStream (wraps another stream and adds mark support)
21              InputStream input3 = new BufferedInputStream(new FileInputStream("example.txt"));
22
23              System.out.println("BufferedInputStream␣supports␣mark:␣" + input3.markSupported()); //
                    true
24
25          }
26          catch (IOException e)
27          {
28              e.printStackTrace();
29          }
30      }
31  }
```

I/O stream XXVIII

Reading data from console by InputStreamReader and BufferedReader: connecting the BufferedReader stream with the InputStreamReader stream for reading the line by line data from the keyboard.

```
1 import java.io.*;
2 public class BufferedReaderExample
3 {
4     public static void main(String args[])throws Exception
5     {
6         InputStreamReader r=new InputStreamReader(System.in);
7         BufferedReader br=new BufferedReader(r);
8         System.out.println("Enter␣your␣name");
9         String name=br.readLine();
10        System.out.println("Welcome␣"+name);
11    }
12 }
```

Reading data from console until user writes stop

```
1 //reading and printing the data until the user prints stop.
2 import java.io.*;
3 public class BufferedReaderExample
4 {
5     public static void main(String args[])throws Exception
6     {
7         InputStreamReader r=new InputStreamReader(System.in);
8         BufferedReader br=new BufferedReader(r);
9         String name="";
10        while(!name.equals("stop"))
11        {
12            System.out.println("Enter␣data:␣");
13            name=br.readLine();
14            System.out.println("data␣is:␣"+name);
15        }
16        br.close();
17        r.close();
18    }
```

```
19 }
```

I/O stream XXXI

CharArrayReader: 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.

```
1import java.io.CharArrayReader;
2public class CharArrayExample
3{
4    public static void main(String[] ag) throws Exception
5    {
6        char[] ary =
7        {
8            'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't'
9        }
10       ;
11       CharArrayReader reader = new CharArrayReader(ary);
12       int k = 0;
13       // Read until the end of a file
14       while ((k = reader.read()) != -1)
15       {
```

I/O stream XXXII

```
16                  char ch = (char) k;
17                  System.out.print(ch + "␣:␣");
18                  System.out.println(k);
19              }
20          }
21      }
```

I/O stream XXXIII

StringReader: Java StringReader class is a character stream with string as a source. It takes an input string and changes it into character stream. It inherits Reader class.

```
1import java.io.StringReader;
2public class StringReaderExample
3{
4    public static void main(String[] args) throws Exception
5    {
6        String srg = "Hello␣Java!!␣\nWelcome␣to␣Javatpoint.";
7        StringReader reader = new StringReader(srg);
8        int k=0;
9        while((k=reader.read())!=-1)
10       {
11           System.out.print((char)k);
12       }
13   }
14}
```

I/O stream XXXIV

FileReader and FileWriter: 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.

```
1import java.io.File;
2import java.io.FileReader;
3import java.io.FileWriter;
4import java.io.IOException;
5public class IOStreamsExample
6{
7    public static void main(String args[]) throws IOException
8    {
9        //Creating FileReader object
10       File file = new File("D:/myFile.txt");
11       FileReader reader = new FileReader(file);
12       char chars[] = new char[(int) file.length()];
13       //Reading data from the file
```

I/O stream XXXV

```
14            reader.read(chars);
15            //Writing data to another file
16            File out = new File("D:/CopyOfmyFile.txt");
17            FileWriter writer = new FileWriter(out);
18            //Writing data to the file
19            writer.write(chars);
20            writer.flush();
21            System.out.println("Data␣successfully␣written␣in␣the␣specified␣file");
22        }
23}
```

InputStreamReader and InputStreamWriter

InputStreamReader: An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Constructor name	Description
InputStreamReader(InputStream in)	It creates an InputStreamReader that uses the default charset.
InputStreamReader(InputStream in, Charset cs)	It creates an InputStreamReader that uses the given charset.
InputStreamReader(InputStream in, CharsetDecoder dec)	It creates an InputStreamReader that uses the given charset decoder.
InputStreamReader(InputStream in, String charsetName)	It creates an InputStreamReader that uses the named charset.

Table 8: Constructor

I/O stream XXXVII

Modifier and Type	Method	Description
void	close()	It closes the stream and releases any system resources associated with it.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
int	read()	It reads a single character.
int	read(char[] cbuf, int offset, int length)	It reads characters into a portion of an array .
boolean	ready()	It tells whether this stream is ready to be read.

Table 9: Method

I/O stream XXXVIII

```
1public class InputStreamReaderExample
2{
3    public static void main(String[] args)
4    {
5        try
6        {
7            InputStream stream = new FileInputStream("file.txt");
8            Reader reader = new InputStreamReader(stream);
9            int data = reader.read();
10           while (data != -1)
11           {
12               System.out.print((char) data);
13               data = reader.read();
14           }
15       }
16       catch (Exception e)
17       {
18           e.printStackTrace();
19       }
```

```
20        }
21}
```

OutputStreamWriter: OutputStreamWriter is a class which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset. write() method calls the encoding converter which converts the character into bytes. The resulting bytes are then accumulated in a buffer before being written into the underlying output stream. The characters passed to write() methods are not buffered. We optimize the performance of OutputStreamWriter by using it with in a BufferedWriter so that to avoid frequent converter invocation.

Constructor name	Description
OutputStreamWriter(OutputStream out)	It creates an OutputStreamWriter that uses the default character encoding.
OutputStreamWriter(OutputStream out, Charset cs)	It creates an OutputStreamWriter that uses the given charset.
OutputStreamWriter(OutputStream out, CharsetEncoder enc)	It creates an OutputStreamWriter that uses the given charset encoder.
OutputStreamWriter(OutputStream out, String charsetName)	It creates an OutputStreamWriter that uses the named charset.

Table 10: Constructor

Modifier and Type	Method	Description
void	close()	It closes the stream, flushing it first.
void	flush()	It flushes the stream.
String	getEncoding()	It returns the name of the character encoding being used by this stream.
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, int off, int len)	It writes a portion of a string.

Table 11: Method

I/O stream XLI

```
1 public class OutputStreamWriterExample
2 {
3     public static void main(String[] args)
4     {
5
6         try
7         {
8             OutputStream outputStream = new FileOutputStream("output.txt");
9             Writer outputStreamWriter = new OutputStreamWriter(outputStream);
10
11            outputStreamWriter.write("Hello World");
12
13            outputStreamWriter.close();
14        }
15        catch (Exception e)
16        {
17            e.getMessage();
18        }
19    }
20 }
```

BufferedInputStream & BufferedOutputStream

BufferedInputStream: 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 declaration

```
1 public class BufferedInputStream extends FilterInputStream
```

Java BufferedInputStream class constructors

Constructor name	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.
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 and reset methods.

I/O stream XLV

```
1import java.io.*;
2public class BufferedInputStreamExample
3{
4    public static void main(String args[])
5    {
6        try
7        {
8            FileInputStream fin=new FileInputStream("D:\\testout.txt");
9            BufferedInputStream bin=new BufferedInputStream(fin);
10           int i;
11           while((i=bin.read())!=-1)
12           {
13               System.out.print((char)i);
14           }
15           bin.close();
16           fin.close();
17       }
18       catch(Exception e)
19       {
```

```
20                  System.out.println(e);
21              }
22          }
23      }
```

I/O stream XLVII

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.
- ✓ The java.io.Console class is attached with system console internally.

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.
PrintWriter writer()	It is used to retrieve the PrintWriterobject associated with the console.
void flush()	It is used to flushes the console.

How to get the object of Console: System class provides a static method console() that returns the singleton instance of Console class.

```
public static Console console()
{

}
```

Example:

```
import java.io.Console;
class ReadStringTest
{
    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);
```

```
10          }
11 }
```

Java Console Example to read password:

```
1import java.io.Console;
2class ReadPasswordTest
3{
4    public static void main(String args[])
5    {
6        Console c=System.console();
7        System.out.println("Enter␣password:␣");
8        char[] ch=c.readPassword();
9        String pass=String.valueOf(ch); //converting char array into string
10       System.out.println("Password␣is:␣"+pass);
11   }
12}
```

DataInputStream & DataOutputStream

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.

DataInputStream class declaration

```
1public class DataInputStream extends FilterInputStream implements DataInput
```

I/O stream LII

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, 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.

I/O stream LIII

```
1import java.io.*;
2public class DataStreamExample
3{
4       public static void main(String[] args) throws IOException
5       {
6               InputStream input = new FileInputStream("testout.txt");
7               DataInputStream inst = new DataInputStream(input);
8               int count = input.available();
9               byte[] ary = new byte[count];
10              inst.read(ary);
11              for (byte bt : ary)
12              {
13                      char k = (char) bt;
14                      System.out.print(k+"—");
15              }
16      }
17}
```

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.

DataOutputStream class declaration

```
public class DataOutputStream extends FilterOutputStream implements DataOutput
```

DataOutputStream class methods

I/O stream LV

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 {
4     public static void main(String[] args) throws IOException
5     {
6         FileOutputStream file = new FileOutputStream(D:\\testout.txt);
7         DataOutputStream data = new DataOutputStream(file);
8         data.writeInt(65);
9         data.flush();
10        data.close();
11        System.out.println("Succcess...");
12    }
13 }
```

RandomAccessFile Class seek() method

- seek() method is available in java.io package.
- seek() method is used to sets the file pointer position calculated from the starting of this file at which the next read or write operation occurs and the position might be set beyond the file.
- seek() method is a non-static method, it is accessible with the class object only and if we try to access the method with the class name then we will get an error.
- seek() method may throw an exception at the time of seeking file pointer.
- IOException: This exception may throw an exception while performing input/output operation.

Constructor	Description
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Table 12: Constructors

Modifier and Type	Method	Method
void	close()	It closes this random access file stream and releases any system resources associated with the stream.
FileChannel	getChannel()	It returns the unique FileChannel object associated with this file.
int	readInt()	It reads a signed 32-bit integer from this file.
String	readUTF()	It reads in a string from this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	writeDouble(double v)	It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.

Table 13: Methods

Modifier and Type	Method	Method
void	writeFloat(float v)	It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	write(int b)	It writes the specified byte to this file.
int	read()	It reads a byte of data from this file.
long	length()	It returns the length of this file.
void	seek(long pos)	It sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

Table 14: Methods

I/O stream LXI

```
1// Java program to demonstrate the example of void seek(long pos) method of RandomAccessFile
2import java.io.*;
3class RAFSeek
4{
5    public static void main(String[] args) throws Exception
6    {
7        // Instantiate a random access file object with file name and permissions
8        RandomAccessFile ra_f = new RandomAccessFile("includehelp.txt", "rw");
9        // By using writeUTF() method is to write data to the file
10       ra_f.writeUTF("Welcome␣in␣Java␣World!!!");
11       // by using seek() method is to set the current file indicator from where read/write could start i.e.
          we set here 0 so reading will be done from 0 till EOF
12       ra_f.seek(0);
13       // By using readUTF() method is to read a data in a string from this file
14       //String str = ra_f.readUTF();
15       System.out.println("ra_f.readUTF():␣" + ra_f.readUTF());
16       // By using close() method is to close this stream ran_f
17       ra_f.close();
18   }
```

I/O stream LXII

```
19 }
```

I/O stream LXIII

Another Example:

```
1 import java.io.IOException;
2 import java.io.RandomAccessFile;
3
4 public class RandomAccessFileExample
5 {
6     static final String FILEPATH ="myFile.TXT";
7     public static void main(String[] args)
8     {
9         try
10        {
11            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
12            writeToFile(FILEPATH, "I␣love␣my␣country␣and␣my␣people", 31);
13        }
14        catch (IOException e)
15        {
16            e.printStackTrace();
17        }
18    }
```

```
19      private static byte[] readFromFile(String filePath, int position, int size)
20      throws IOException
21      {
22          RandomAccessFile file = new RandomAccessFile(filePath, "r");
23          file.seek(position);
24          byte[] bytes = new byte[size];
25          file.read(bytes);
26          file.close();
27          return bytes;
28      }
29      private static void writeToFile(String filePath, String data, int position)
30      throws IOException
31      {
32          RandomAccessFile file = new RandomAccessFile(filePath, "rw");
33          file.seek(position);
34          file.write(data.getBytes());
35          file.close();
36      }
37  }
```