INPUTSTREAM AND OUTPUTSTREAMS

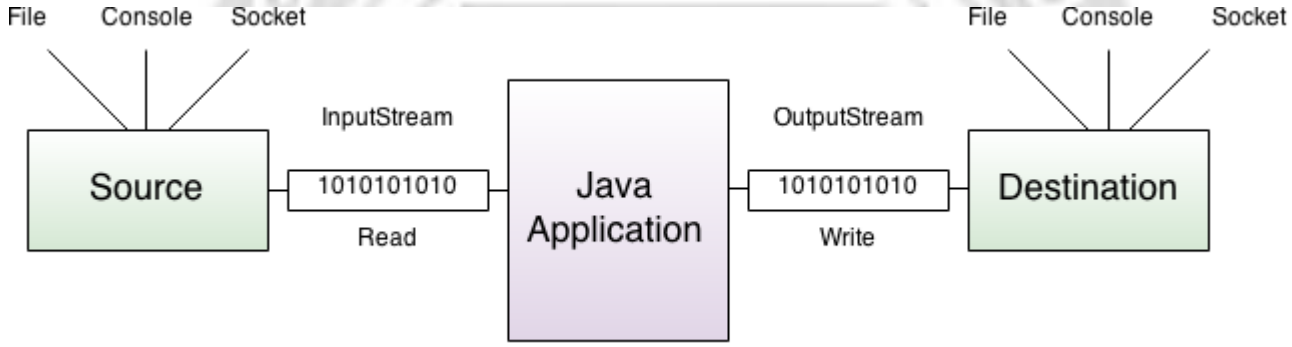
✓ **OutputStream**

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

✓ **InputStream**

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

Working of Java OutputStream and InputStream by the figure given below.



OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Commonly used methods of OutputStream class

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

InputStream class

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

Commonly used methods of InputStream class

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

1. FileInputStream and FileOutputStream (File Handling):

In Java, FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

✓ FileOutputStream class

Java FileOutputStream is an output stream for writing data to a file.

If you have to write primitive values then use FileOutputStream. Instead, for character-oriented data, prefer FileWriter. But you can write byte-oriented as well as character-oriented data.

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

Example of Java FileOutputStream class

```
1. import java.io.*;
2. class Test{
3.   public static void main(String args[]){
4.    try{
5.      FileOutputstream fout=new FileOutputStream("abc.txt");
6.      String s="java is my favourite language";
7.      byte b[]=s.getBytes();//converting string into byte array
8.      fout.write(b);
9.      fout.close();
```

```
10.    System.out.println("success...");
11.   }catch(Exception e){system.out.println(e);}
12. }
13.}
```

Output:success...

✓ **FileInputStream class**

Java FileInputStream class obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.

It should be used to read byte-oriented data for example to read image, audio, video etc.

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

Example of FileInputStream class

```
1.  import java.io.*;
2.  class SimpleRead{
3.   public static void main(String args[]){
4.    try{
5.     FileInputStream fin=new FileInputStream("abc.txt");
6.     int i=0;
7.     while((i=fin.read())!=-1){
8.     System.out.println((char)i);
9.     }
10.    fin.close();
11.  }catch(Exception e){system.out.println(e);}
12. }
13.}
```

Output: java is my favourite language

2. BufferedOutputStream and BufferedInputStream

✓ BufferedOutputStream class

Java BufferedOutputStream class uses an internal buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.

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

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

Example of BufferedOutputStream class:

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

```
1. import java.io.*;
2. class Test{
3.  public static void main(String args[])throws Exception{
4.    FileOutputStream fout=new FileOutputStream("f1.txt");
5.    BufferedOutputStream bout=new BufferedOutputStream(fout);
6.    String s="Java is my favourite language";
7.    byte b[]=s.getBytes();
8.    bout.write(b);
9.    bout.flush();
10.   bout.close();
11.   fout.close();
```

```
12.  System.out.println("success");
13. }
14.}
```

Output: success...

✓ **BufferedInputStream class**

Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

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

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.

Example of Java BufferedInputStream

```
1. import java.io.*;
2. class SimpleRead{
3.  public static void main(String args[]){
4.   try{
```

```
5.  FileInputStream fin=new FileInputStream("f1.txt");
6.  BufferedInputStream bin=new BufferedInputStream(fin);
7.  int i;



8.  while((i=bin.read())!=-1){
9.   System.out.println((char)i);
10. }
11. bin.close();
12. fin.close();
13. }catch(Exception e){system.out.println(e);}
14. }
15.}
```

Output: Java is my favourite language

3. DataInputStream and DataOutputStream:

✓ DataInputStream class

DataInputStream class allows the programmer to read primitive data from the input source.

Method	Description
int read(byte[] b)	It is used to read the number of bytes 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.
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.

✓ DataOutputStream class

The DataOutputStream stream let you write the primitives to an output source.

Example:

Following is the example to demonstrate DataInputStream and DataInputStream. This example reads 5 lines given in a file test.txt and converts those lines into capital letters and finally copies them into another file test1.txt.

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 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 flush()	It is used to flushes the data output stream.

Test.txt

this is test 1 ,
this is test 2 ,
this is test 3 ,
this is test 4 ,
this is test 5 ,

test.java

```
import java.io.*;
public class Test{
  public static void main(String args[])throws IOException{
    DataInputStream d = new DataInputStream(new FileInputStream("test.txt"));
    DataOutputStream out = new DataOutputStream(new FileOutputStream("test1.txt"));
    String count;
    while((count = d.readLine()) != null){
      String u = count.toUpperCase();
      System.out.println(u);
      out.writeBytes(u + "  ,");}
```

```
    d.close();
    out.close();
  }}
```

Output:

THIS IS TEST 1 ,
THIS IS TEST 2 ,
THIS IS TEST 3 ,
THIS IS TEST 4 ,
THIS IS TEST 5 ,

4. PrintStream

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.

Commonly used methods of PrintStream class:

There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:

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

the line./li>

- **public void println(Object obj):** it prints the specified object value and terminates the line.
- **public void println():** it terminates the line only.
- **public void printf(Object format, Object... args):** it writes the formatted string to the current stream.
- **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream.

- **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format.
- **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format.

Example of java.io.PrintStream class:

In this example, we are simply printing integer and string values.

```
1. import java.io.*;
2. class PrintStreamTest{
3.  public static void main(String args[])throws Exception{
4.    FileOutputStream fout=new FileOutputStream("mfile.txt");
5.    PrintStream pout=new PrintStream(fout);
6.    pout.println(1900);
7.    pout.println("Hello Java");
8.    pout.println("Welcome to Java");
9.    pout.close();
10.   fout.close();
11. }
12.}
```

Example of printf() method of java.io.PrintStream class:

Example of printing integer value by format specifier:

```
1. class PrintStreamTest{
2.  public static void main(String args[]){
3.    int a=10;
4.  System.out.printf("%d",a);//Note, out is the object of PrintStream class
5.  }
6. }
```

Output:10

➢ CHARACTER STREAMS (READER & WRITER):

Java IO's Reader and Writer work much like the InputStream and OutputStream with the exception that Reader and Writer are character based. They are intended for reading and writing text. The InputStream and OutputStream are byte based.

Reader class:

The **Java.io.Writer** class is a abstract class for writing to character streams.

Methods defined by Reader class:

Method	Description
abstract void close()	This method closes the stream and releases any system resources associated with it.
void mark(int numChars)	This method marks the present position in the stream.
boolean markSupported()	This method tells whether this stream supports the mark() operation.
int read()	This method reads a single character.
int read(char buffer[])	This method reads characters into an array.
abstract int read(char buffer[],int offset,int numChars)	This method reads characters into a portion of an array.
boolean ready()	This method tells whether this stream is ready to be read.
void reset()	This method resets the stream.
long skip(long numChars)	This method skips characters.

Writer class:

The Java.io.Writer class is a abstract class for writing to character streams

Methods defined by Writer class:

Method	Description
Writer append(char ch)	This method appends the specified character to this writer.
Writer append(CharSequence chars)	This method appends the specified character sequence to this writer.
Writer append(CharSequence chars, int begin, int end)	This method appends the specified character sequence to this writer.
abstract void close()	This method loses the stream, flushing it first.
abstract void flush()	This method flushes the stream.

void write(int ch)	This method writes a single character.
void write(char buffer[])	This method writes an array of characters.

1. Java FileWriter and FileReader (File Handling in java)

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

➢ Java FileWriter class

Java FileWriter class is used to write character-oriented data to the file.
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
1) public void write(String text)	writes the string into FileWriter.
2) public void write(char c)	writes the char into FileWriter.
3) public void write(char[] c)	writes char array into FileWriter.
4) public void flush()	flushes the data of FileWriter.
5) public void close()	closes FileWriter.

➢ Java FileReader class

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

Constructors of FileWriter 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
public int read()	returns a character in ASCII form. It returns -1 at the end of file.
public void close()	closes FileReader.

2. BufferedReader and BufferedWriter classes:

- **BufferedWriter class:**

  This can be used for writing character data to the file.

Constructors

BufferedWriter bw = new BufferedWriter(writer w)
BufferedWriter bw = new BufferedWriter(writer r, int size)

BufferedWriter never communicates directly with the file. It should be communicate through some writer object only.

Important methods of BufferedWriter Class

void write(int ch) thorows IOException
void write(String s) throws IOException
void write(char[] ch) throws IOException
void newLine() for inserting a new line character.
void flush()
void close()

- **BufferedReader**

  BufferedReader class can read character data from the file.

Constructors

1. BufferedReader br = new BufferedReader(Reader r)
2. BufferedReader br = new BufferedReader(Reader r, int buffersize)
3. BufferedReader never communicates directly with the file. It should Communicate through some reader object only.

Important methods of BufferedReader Class

1. int read()
2. int read(char [] ch)
3. String readLine(); - Reads the next line present in the file. If there is no nextline this method returns null.
4. void close()

Example for Java FileWriter and FileReader , BufferedReader and BufferedWriter classes:

```
import java.io.*;
```

```
class Simple{
public static void main(String args[]){
try{
        FileWriter fw=new FileWriter("d:/archana/abc.txt");
       BufferedWriter bw = new BufferedWriter(fw);
       bw.write(" Java");
        bw.close();
       fw.close();
        FileReader fr=new FileReader("d:/archana/abc.txt");
        BufferedReader br = new BufferedReader(fr);
         int i;
         while((i=br.read())!=-1)




         System.out.print((char)i);
         br.close();
        fr.close();
       }catch(Exception e){System.out.println(e);}
        System.out.println("success");
       } }
```

Output

Java
success

3. InputStreamReader and OutputStreamWriter classes:

- **OutputStreamWriter**

OutputStreamWriter behaves as a bridge to transfer data from character stream to byte stream. It uses default charset or we can specify charset for change in character stream to byte stream.

Constructors

1. OutputStreamWriter(OutputStream out)
2. OutputStreamWriter(OutputStream out, Charset cs)
3. OutputStreamWriter(OutputStream out, CharsetEncoder enc)
4. OutputStreamWriter(OutputStream out, String charsetName)

Important methods of OutputStreamWriter

1. void close()
2. void flush()
3. String getEncoding()
4. void write(int c)
5. void write(String str, int off, int len)

OutputStreamWriterDemo.java

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
```

```
import java.io.Writer;
public class OutputStreamWriterDemo {
        public static void main(String[] args) {
    String str = "Hello World! \nThis is OutputStreamWriter Code Example."
    BufferedWriter bw = null;
    try {
     Writer w = new OutputStreamWriter(System.out);
     bw = new BufferedWriter(w);
     bw.write(str);
    } catch (IOException e) {
      e.printStackTrace();
    }finally {









    try {
          bw.close();
        } catch (IOException ex) {
          ex.printStackTrace();
        }
     }
  }
}
```

Output

Hello World!
This is OutputStreamWriter Code Example.

- **InputStreamReader**

InputStreamReader behaves as bridge from bytes stream to character stream. It also uses charset to decode byte stream into character stream.

Constructors

1. InputStreamReader(InputStream in_strm)
2. InputStreamReader(InputStream in_strm, Charset cs)
3. InputStreamReader(InputStream in_strm, CharsetDecoder dec)
4. InputStreamReader(InputStream in_strm, String charsetName)

Important methods of InputStreamReader

1. public boolean ready() – tells whether the character stream is ready to be read or not.
2. public void close() – closes InputStreamReader and releases all the Streams associated with it.
3. public int read() – returns single character after reading.
4. public String getEncoding() – returns the name of the character encoding being used by this stream.

InputStreamReaderDemo.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class InputStreamReaderDemo {
        public static void main(String[] args) {
                InputStreamReader isr = new InputStreamReader(System.in);
                BufferedReader br = new BufferedReader(isr);
                int a=0;
                int b=0;
                try {
                        System.out.println("Enter a number..");
                        a = Integer.parseInt(br.readLine());
                        System.out.println("Enter another number..");
                         b = Integer.parseInt(br.readLine());
                } catch (NumberFormatException e)




                {
                                e.printStackTrace();
                } catch (IOException e) {
                                e.printStackTrace();
                }
                System.out.println("you entered "+a+" and  "+b); }}
```

Output

```
Enter a number..
10
Enter another number..
14
you entered 10 and  14
```

4. PrintWriter Class

The **Java.io.PrintWriter** class prints formatted representations of objects to a text-output stream.

✓ **PrintWriter** defines several constructors. The one we will use is shown here:

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

Here, *outputStream* is an object of type **OutputStream**, and *flushOnNewline* controls whether Java flushes the output stream every time a newline ('**\\n**') character is output. If *flushOnNewline* is **true**, flushing automatically takes place. If **false**, flushing is not automatic.

✓ **PrintWriter** supports the **print( )** and **println( )** methods for all types including **Object**. Thus, you can use these methods in the same way as they have been used with **System.out**. If an argument is not a simple type, the **PrintWriter** methods call the object's **toString( )** method and then print the result.

✓ To write to the console by using a **PrintWriter**, specify **System.out** for the output stream and flush the stream after each newline. For example, this line of code creates a **PrintWriter** that is connected to console output:

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

The following application illustrates using a **PrintWriter** to handle console output:

```
// Demonstrate PrintWriter
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");
int i = -7;
pw.println(i);
double d = 4.5e-7;
pw.println(d); } }
```

The output from this program is shown here:

```
This is a string
-7
4.5E-7
```

4.2: READING AND WRITING CONSOLE

3 Ways to read input from console in Java

1. Using Buffered Reader Class

Advantages

The input is buffered for efficient reading.

Drawback:

The wrapping code is hard to remember.

2. Using Scanner Class

Advantages:

- Convenient methods for parsing primitives (nextInt(), nextFloat(), ...) from the tokenized input.
- Regular expressions can be used to find tokens.

Drawback:

The reading methods are not synchronized

3. Using Console Class

Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.

Drawback:

Does not work in non-interactive environment (such as in an IDE).

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

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

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.
PrintWriter writer()	It is used to retrieve the PrintWriter object 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(){}

Let's see the code to get the instance of Console class.

Console c=System.console();

Java Console Example

```
1. import java.io.Console;
2. class ReadStringTest{
3. public static void main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter your name: ");
6. String n=c.readLine();
7. System.out.println("Welcome "+n);
8. }
9. }
```

Output

```
Enter your name: abcd
Welcome abcd
```

Java Console Example to read password

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

Output

```
Enter password:
Password is: 123
```

4.3: READING AND WRITING FILES

What is File Handling in Java?

- ✓ File handling in Java implies reading from and writing data to a file.
- ✓ The File class from the **java.io package**, allows us to work with different formats of files.
- ✓ In order to use the File class, you need to create an object of the class and specify the filename or directory name.

For example:

```
1)  // Import the File class
2)  import java.io.File
3)  // Specify the filename
4)  File obj = new File("filename.txt");
```

Java uses the concept of a stream to make I/O operations on a file.

The File class has many useful methods for creating and getting information about files.

For example:

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

➢ File Operations in Java

Basically, you can perform four operations on a file. They are as follows:

1) Create a File
2) Get File Information
3) Write To a File
4) Read from a File

1) Create a File

To create a file in Java, you can use the createNewFile() method. This method returns a boolean value: true if the file was successfully created, and false if the file already exists.

Example:

```
import java.io.File;
import java.io.IOException;

public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

The output will be:

```
File created: filename.txt
```

2) Write To a File

In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above. Note that when we are done writing to the file, we should close it with the close() method:

Example:

```
import java.io.FileWriter;  // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors
```

```
public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Files in Java might be tricky, but it is fun enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Output:

Successfully wrote to the file.

3) Read a File

In the following example, we use the Scanner class to read the contents of the text file we created in the previous example:

Example:

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException; // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files
public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }}}
```

Output:

Files in Java might be tricky, but it is fun enough!

4) Get File Information

To get more information about a file, use any of the File methods:

Example:

```
import java.io.File; // Import the File class
```

```
public class GetFileInfo {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.exists()) {
      System.out.println("File name: " + myObj.getName());
      System.out.println("Absolute path: " + myObj.getAbsolutePath());
      System.out.println("Writeable: " + myObj.canWrite());
      System.out.println("Readable " + myObj.canRead());
      System.out.println("File size in bytes " + myObj.length());
    } else {
      System.out.println("The file does not exist.");
    }
  }
}
```

Output:

```
File name: filename.txt
Absolute path: C:\Users\MyName\filename.txt
Writeable: true
Readable: true
File size in bytes: 0
```