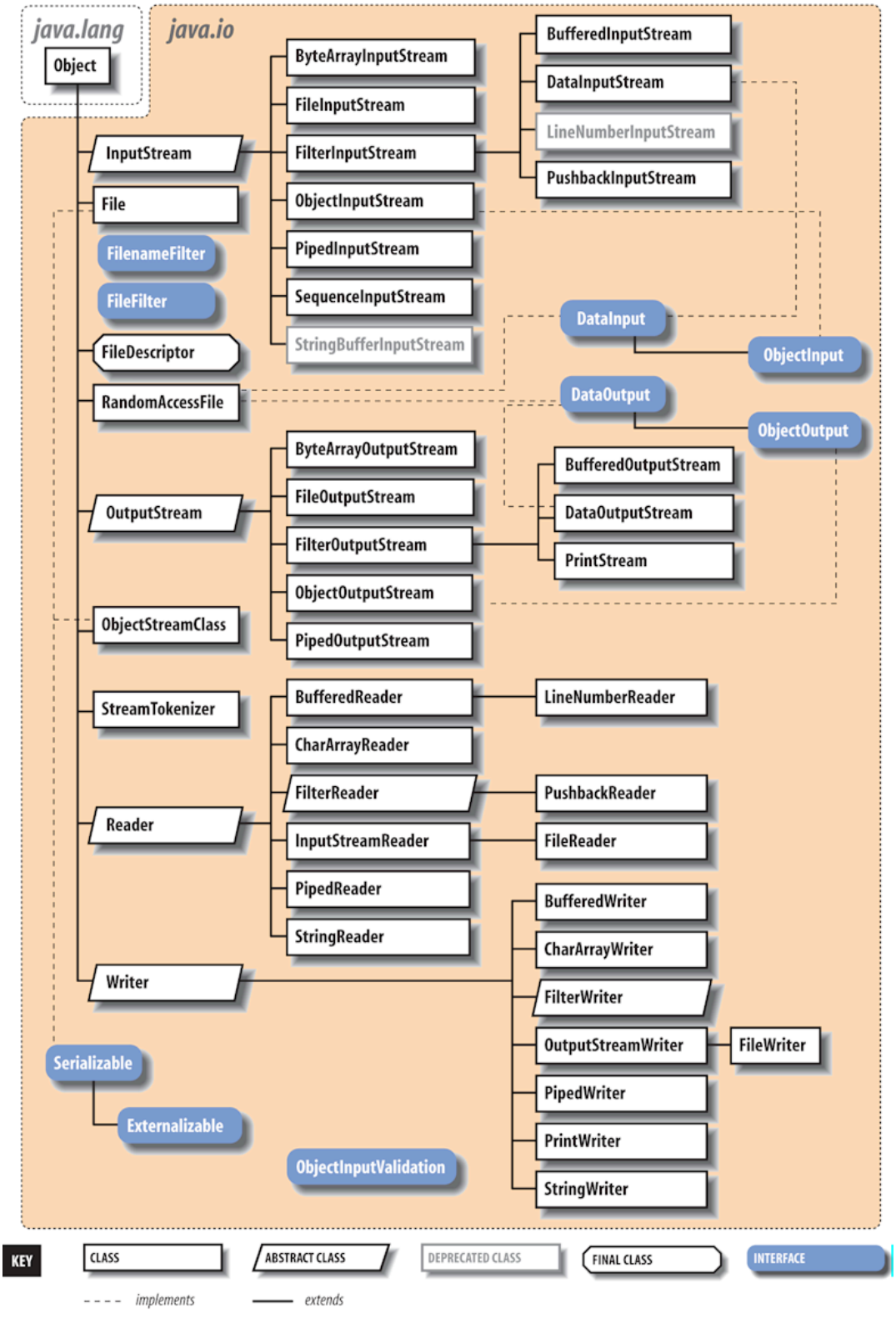
Files and I/O in Java

The java.io package contains nearly every class required to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc. Java provides strong and flexible support for I/O related to files and networks.

The Hierarchy of I/O Class in Java



Stream

A stream can be defined as a sequence of data. There are two kinds of Streams :

- **InputStream** – The InputStream is used to read data from a source.
- **OutputStream** – The OutputStream is used for writing data to a destination.

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**.

Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

Why there are Byte and Character Streams

A stream is a way of sequentially accessing a file. A byte stream access the file byte by byte. A byte stream is suitable for any kind of file, however not quite appropriate for text files. For example, if the file is using a unicode encoding and a character is represented with two bytes, the byte stream will treat these separately and you will need to do the conversion yourself.

A character stream will read a file character by character. A character stream needs to be given the file's encoding in order to work properly.

Standard Streams

All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen. Java provides the following three standard streams

- **Standard Input** − This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.
- **Standard Output** − This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out**.
- **Standard Error** − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err**.

Reading and Writing Files

The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination. The two important streams are **FileInputStream** and **FileOutputStream**.

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file.

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method.

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have InputStream object, You can use various methods :

Sr.No.	Method & Description
	public void close() throws IOException{}
1	This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
	protected void finalize()throws IOException {}
2	This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
	public int read(int r)throws IOException{}
3	This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.

	public int read(byte[] r) throws IOException{}
4	This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
	public int available() throws IOException{}
5	Gives the number of bytes that can be read from this file input stream. Returns an int.

Other important input streams :

- ByteArrayInputStream
- DataInputStream

FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Following constructor takes a file name as a string to create an input stream object to write the file

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method.

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Once you have InputStream object, You can use various methods :

Sr.No.	Method & Description
	public void close() throws IOException{}
1	This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2	**protected void finalize()throws IOException {}**

	This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	**public void write(int w)throws IOException{}** This methods writes the specified byte to the output stream.
4	**public void write(byte[] w)** Writes w.length bytes from the mentioned byte array to the OutputStream.

Other important output streams :

- ByteArrayOutputStream
- DataOutputStream

Complete Example

The below code would create file test.txt and would write given numbers in binary format. Same would be the output on the stdout screen.

```
import java.io.*;
public class fileStreamTest {

   public static void main(String args[]) {

      try {
         byte bWrite [] = {11,21,3,40,5};
         OutputStream os = new FileOutputStream("test.txt");
         for(int x = 0; x < bWrite.length ; x++) {
            os.write( bWrite[x] );   // writes the bytes
         }
         os.close();

         InputStream is = new FileInputStream("test.txt");
         int size = is.available();

         for(int i = 0; i < size; i++) {
            System.out.print((char)is.read() + "  ");
         }
         is.close();
      } catch (IOException e) {
         System.out.print("Exception");
      }
   }
}
```

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. **Example :**

```
import java.io.Console;
class ReadPasswordTest {
    public static void main(String args[]) {
        Console c = System.console();
        System.out.println("Enter password: ");
        char[] ch = c.readPassword();
        String pass = String.valueOf(ch); //converting char array i
        System.out.println("Password is: " + pass);
    }
}
```

File Navigation and I/O

Other classes for File Navigation and I/O.

- File Class
- FileReader Class
- FileWriter Class

Directories in Java

A directory is a File which can contain a list of other files and directories. You use **File** object to create directories, to list down files available in a directory.

Creating Directories

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Below code creates "/tmp/user/java/bin" directory.

```
import java.io.File;
public class CreateDir {

   public static void main(String args[]) {
      String dirname = "/tmp/user/java/bin";
      File d = new File(dirname);

      // Create directory now.
      d.mkdirs();
   }
}
```

NOTE : Java automatically takes care of path separators on UNIX and Windows as per conventions. If you use a forward slash (/) on a Windows version of Java, the path will still resolve correctly.

Listing Directories

You can use list( ) method provided by File object to list down all the files and directories available in a directory as follows.

```java
import java.io.File;
public class ReadDir {

   public static void main(String[] args) {
      File file = null;
      String[] paths;

      try {
         // create new file object
         file = new File("/tmp");

         // array of files and directory
         paths = file.list();

         // for each name in the path array
         for(String path:paths) {
            // prints filename and directory name
            System.out.println(path);
         }
      } catch (Exception e) {
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```