Chapter 6 : Strings, Streams and Files

String : string is basically an object that represents sequence of char values. An array of characters works same as Java string.

- **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
- String s=**new** String(ch);
- is same as:
- String s="javatpoint";
- **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

StringBuffer class in Java

StringBuffer class is used to create a **mutable** string object. It means, it can be changed after it is created. It represents growable and writable character sequence.

It is similar to String class in Java both are used to create string, but stringbuffer object can be changed.

So **StringBuffer** class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors.

StringBuffer(): It creates an empty string buffer and reserves space for 16 characters.

StringBuffer(int size): It creates an empty string and takes an integer argument to set capacity of the buffer.

StringBuffer(String str): It creates a stringbuffer object from the specified string.

StringBuffer(charSequence []ch): It creates a stringbuffer object from the charsequence array.

<u>**Creating a StringBuffer Object**</u> **:**

```
public class Demo
{
 public static void main(String[] args) {
StringBuffer sb = new StringBuffer("study");
System.out.println(sb);            // modifying object sb.append("tonight");
System.out.println(sb);              // Output: studytonight
}
}
```

Difference between String and StringBuffer

In this example, we are creating objects of String and StringBuffer class and modifying them, but only stringbuffer object get modified. See the below example.

```
class Test {
public static void main(String args[])
{
String str = “Modern";
str.concat(“Codelab");
System.out.println(str);              // Output: Modern
```

```
  StringBuffer strB = new StringBuffer(“Modern");
strB.append(“Codelab");
System.out.println(strB);          // Output: ModernCodelab
}
}
```

Important methods of StringBuffer class

The following methods are some most commonly used methods of StringBuffer class.

append()

This method will concatenate the string representation of any type of data to the end of the StringBuffer object. append() method has several overloaded forms.

- StringBuffer append(String str)
- StringBuffer append(int n)
- StringBuffer append(Object obj)

The string representation of each parameter is appended to **StringBuffer** object.

```
public class Demo {
          public static void main(String[] args) {
                    StringBuffer str = new StringBuffer("test");
                    str.append(123);
                    System.out.println(str);
          }
}
```

Insert()

This method inserts one string into another. Here are few forms of insert() method.

- StringBuffer insert(int index, String str)
- StringBuffer insert(int index, int num)
- StringBuffer insert(int index, Object obj)

Here the first parameter gives the index at which position the string will be inserted and string representation of second parameter is inserted into **StringBuffer** object.

reverse()

This method reverses the characters within a **StringBuffer** object.

```
public class Demo {
        public static void main(String[] args) {
                StringBuffer str = new StringBuffer("Hello");
                str.reverse();
                System.out.println(str);
        }
}
```

replace()

This method replaces the string from specified start index to the end index.

```
public class Demo {
        public static void main(String[] args) {
                StringBuffer str = new StringBuffer("Hello World");
                str.replace( 6, 11, "java");
                System.out.println(str);
        }
}
```

Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

The format() method of java language is like *sprintf()* function in c language and *printf()* method of java language.

Internal implementation

```
public static String format(String format, Object... args)
{
   return new Formatter().format(format, args).toString();
  }
```

Java String format() method example

Java String format() method example

```
public class FormatExample{
public static void main(String args[]){
String name="sonoo";
String sf1=String.format("name is %s",name);
String sf2=String.format("value is %f",32.33434);
String sf3=String.format("value is %32.12f",32.33434);
//returns 12 char fractional part filling with 0
System.out.println(sf1);
System.out.println(sf2);
System.out.println(sf3);
}}
```

What is a Stream?

Java provides I/O Streams to read and write data where, a Stream represents an input source or an output destination which could be a file, i/o devise, other program etc.

In general, a Stream will be an input stream or, an output stream.

InputStream − This is used to read data from a source.

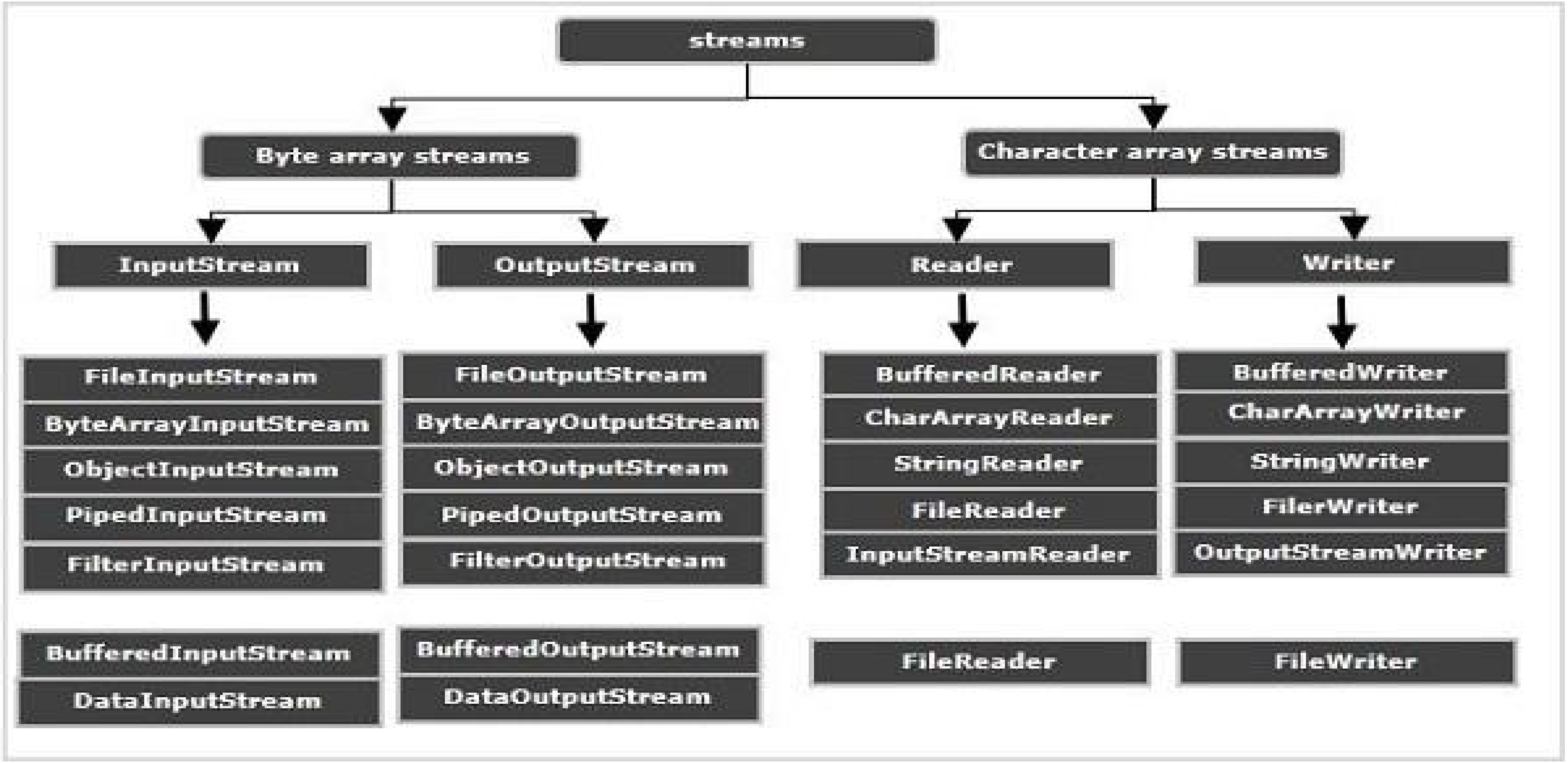
OutputStream − This is used to write data to a destination.

Based on the data they handle there are two types of streams −

Byte Streams − These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can store characters, videos, audios, images etc.

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

Following diagram illustrates all the input and output Streams (classes) in Java.



Standard Streams

In addition to above mentioned classes Java provides 3 standard streams representing the input and, output devices.

Standard Input − This is used to read data from user through input devices. keyboard is used as standard input stream and represented as System.in.

Standard Output − This is used to project data (results) to the user through output devices. 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.

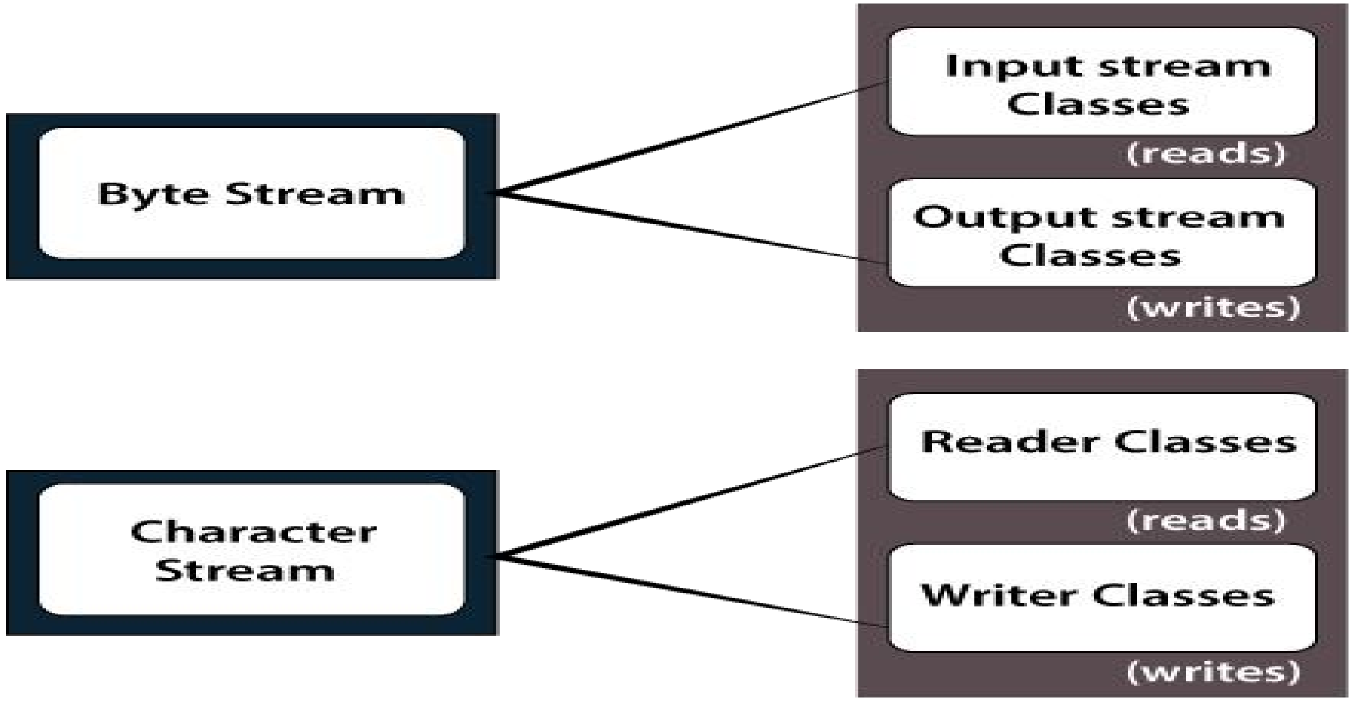
File Operations in Java

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**.

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

Byte Stream

Byte Stream is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.

Character Stream

Character Stream is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

File Operations

We can perform the following operation on a file :

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

Create a 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.

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.

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.

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.

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.

Random Access File

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Constructor

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.