BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2202

JAVA PROGRAMMING

Unit 7

Streams in Java

Table of Contents

1. INTRODUCTION

In the last unit, we discussed handling exceptions in Java. In this unit, we will discuss various streams in Java. All programs accept input from the user, process it, and produce an output. Therefore, all programming languages support input and output operations. Java provides support for input and output through the classes of the ***java.io*** package.

1.1 Objectives:

After studying this unit, you will be able to:

- *Discuss streams basics*
- *Explain abstract streams*
- *Discuss various stream classes*
- *Explain Readers and Writers*
- *Discuss Random Access Files*
- *Explain the concept of Serialization*
- *Explain the concept of streams in the Java Collections API*

2. STREAMS BASICS

Java handles all input and output in the form of streams. A stream is a sequence of bytes traveling from a source to a destination over a communication path. If a program writes into a stream, it is the stream's source. Similarly, if it is reading from a stream, it is the stream's destination. Streams are powerful because they abstract the details of input/output (I/O) operations.

The two basic streams used are the **input** and the **output** streams. Each stream has a particular functionality. You can only read from an input stream and conversely, you can only write to an output stream.

Some streams are classified as ***mode streams*** because they read from or write to a specific place like a disk file or memory. Some streams are classified as ***filters***. **Filters** are used to read data from one stream and write it to another stream.

The stream classes of java.io packages are categorized into two groups based on the data type on which they operate:

- **Byte Stream:** Provide support for handling I/O on bytes.
- **Character stream:** Provide support for handling I/O on characters.

The two major classes for byte streams are **InputStream** and **OutputStream**. These abstract classes are sub-classed to provide a variety of I/O capabilities. The InputStream class lays the foundation for the input class hierarchy, whereas the OutputStream class lays the foundation for the output class hierarchy.

The diagrams given in the figure 7.1 and figure 7.2 represent the hierarchy of the two streams.

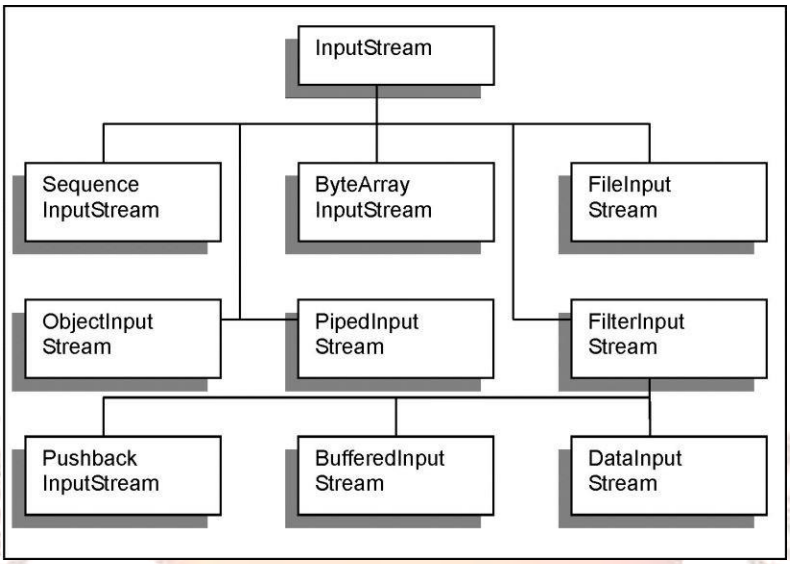

Figure 7.1: InputStream Class Hierarchy

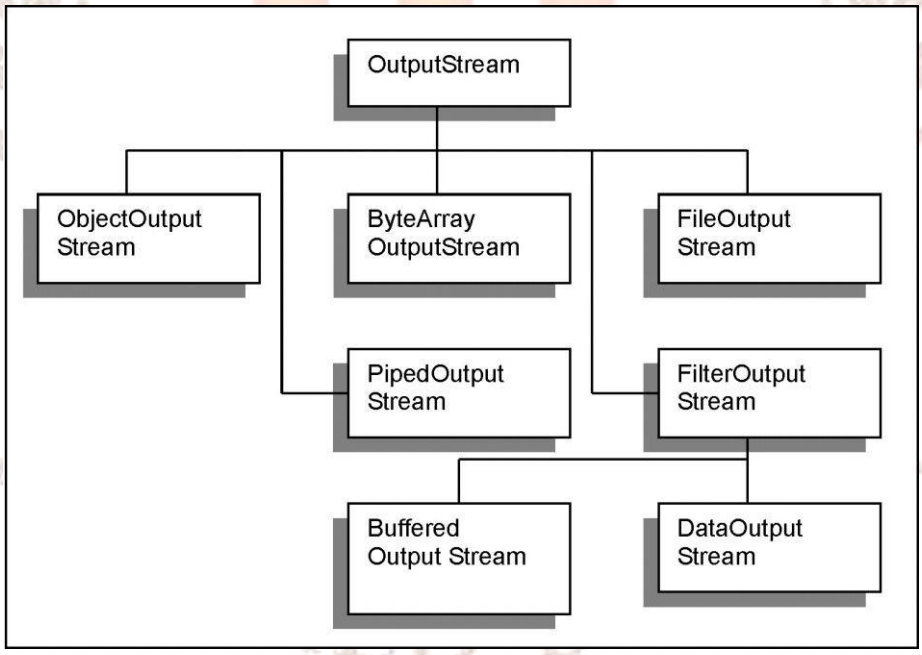

Figure 7.2: OutputStream Class Hierarchy

JDK 1.1 introduces the **Reader** and the **Writer** classes for Unicode I/O. These classes support internationalization. In the figure 7.3 the hierarchy of reader classes and in the figure 7.4 the hierarchy of writer classes is given.

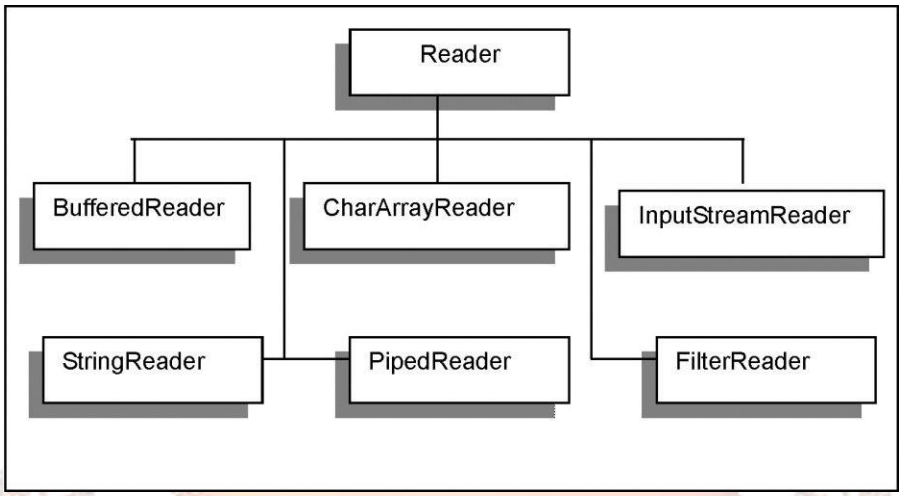

Figure 7.3: Reader Class Hierarchy

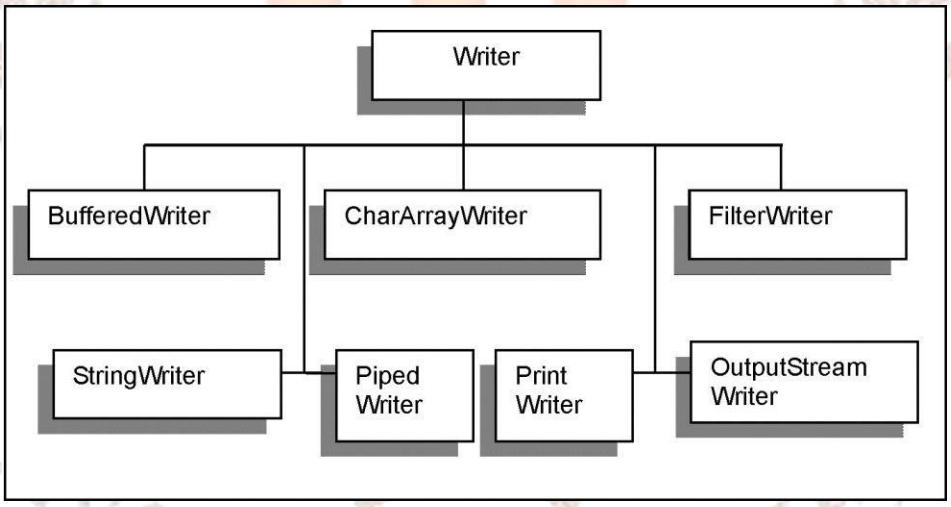

Figure 7.4: Writer Class Hierarchy

Self-Assessment Questions - 1

1. Java handles all input and output in the form of _____ .
2. The two basic streams used are the _____ and the _____ streams.
3. _____ are used to read data from one stream and write it to another stream.

3. THE ABSTRACT STREAMS

The methods available in InputStream class are shown in Table 7.1.

Table 7.1: Methods of the InputStream Class

Method	Explanation
int read ()	Reads a byte of data from the input stream.
int read (byte [] b)	Reads bytes of data from the input stream and stores themin the array.
int read (byte [] b, int off, int len)	Reads the number of bytes specified by the third argument from the input stream and stores it in the array. The second argument specifies the position from where the bytes have to be read.
int available ()	Returns the number of bytes available for reading from the input stream.
void close ()	Close an input stream and releases all the resources associated with it.

Markable Streams

Markable streams provide the capability of placing "bookmarks" on the stream and resetting it. This stream can then be read from the marked position. If a stream is marked, it must have some memory associated with it to keep track of the data between the mark and the current position of the stream. Table 7.2 shows the various markable stream methods and table 7.3 shows various methods available in OutputStream class.

Table 7.2: Markable Stream Methods

Method	Explanation
boolean markSupported ()	Returns true if the stream supports a bookmark.
void mark (int readlimit)	Marks a position on the stream and identifies the number of bytes that can be read before the mark becomes invalid.
void reset ()	Repositions the stream to its last marked position.
long skip (long n)	Skips a specific number of bytes in the stream.

Table 7.3: Methods of the OutputStream Class

Method	Explanation
void write (int n)	Writes the specified byte of data to the output stream.
void write (byte [] b)	Writes an array of bytes to the output stream.
void write (byte [] b,int off, int len)	Writes a segment of an array to the output streams.
void flush ()	Force writes whenever the data accumulates in theoutput stream.
void close ()	Cause the output stream to close. Any data present on itis stored before the stream is de-allocated from thememory.

Self-Assessment Questions - 2

4. _____ provide the capability of placing "bookmarks" on the stream and resetting it.
5. _____ method writes the specified byte of data to the output stream.

4. STREAM CLASSES

Stream Classes are classified as FileInputStream, FileOutputStream, BufferedInputStream, BufferedOutputStream, DataInputStream, and DataOutputStream classes.

The *FileInputStream* and *FileOutputStream* Classes

These streams are classified as ***mode streams*** as they read and write data from disk files. The classes associated with these streams have constructors that allow you to specify the path of the file to which they are connected. The **FileInputStream** class allows you to read input from a file in the form of a stream. The **FileOutputStream** class allows you to write output to a file stream.

Example:

FileInputStream inputfile = new FileInputStream ("Employee.dat");

FileOutputStream outputfile = new FileOutputStream ("binus.dat");

The BufferedInputStream and BufferedOutputStream Classes

The **BufferedInputStream** class creates and maintains a buffer for an input stream. This class is used to increase the efficiency of input operations. This is done by reading data from the stream one byte at a time. The **BufferedOutputStream** class creates and maintains a buffer for the output stream. Both the classes represent filter streams.

The DataInputStream and DataOutputStream Classes

The **DataInputStream** and **DataOutputStream** classes are the filter streams that allow the reading and writing of Java primitive data types.

The **DataInputStream** class provides the capability to read primitive data types from an input stream. It implements the methods presents in the Data Input interface. The methods of **DataInputStream** are listed in the table 7.4.

Table 7.4: Methods of the DataInputStream class

Method	Explanation
boolean readBoolean ()	Reads one byte and returns true if that byte isnonzero, false if it is zero.
byte readByte ()	Reads a byte as an 8-bit signed value.
char readChar ()	Reads a Unicode character.
int readInt ()	Reads an integer value.

The DataOutputStream class provides methods that are complementary to the methods of DataInputStream classes. It keeps track of the number of bytes written to the output stream. Table 7.5 lists the methods of the DataOutputStream class.

Table 7.5: Methods of the DataOutputStream Class

Method	Explanation
void writeChar (int v)	Writes a character to the output stream.
void writeLong (long v)	Writes a long integer to the output stream.
void writeInt (int v)	Writes an integer to the output stream.

The *ByteArrayInputStream* and *ByteArrayOutputStream* Classes

These classes allow you to read and write an array of bytes. In the table 7.6, the methods of ByteArrayInputStream is listed.

Table 7.6: Methods of ByteArrayInputStream Classes

Method	Explanation
ByteArrayInputStream (byte [] b)	Creates an input stream of the array of byes
int size ()	Returns the total number of bytes written to the stream.

Self-Assessment Questions - 3

6. _____ and_____ streams are classified as mode streams as they read and write data from disk files.
7. The _____ and _____ classes are the filter streams that allow the

5. READERS AND WRITERS

The subclasses of the input and output streams read and write data in the form of bytes. The **Reader** and **Writer** classes are abstract classes that support the reading and writing of Unicode character streams. **Unicode** is used to represents data such that each character is represented by **16 bits**. Byte streams work on eight bits of data. These 16-bit version streams are called Reader and Writer.

The most important subclasses of the Reader and Writer classes are **InputStreamReader** and **OutputStreamWriter**. These classes are used as interfaces between byte streams and character readers and writers.

The Reader class supports InputStream class methods like read(), skip(), mark(), markSupported(), reset() and close().

The writer class supports methods of the OutputStream class like write(), flush() and close().

Reading from the Keyboard

The **InputStreamReader** and **OutputStreamReader** classes are used to convert data between the byte and Unicode character streams. The InputStreamReader class converts an InputStream subclass object into Unicode character stream. The OutputStreamWriter class converts a Unicode character output stream into a byte output stream.

Buffered character I/O is supported by the **BufferedReader** and **BufferedWriter** classes. The ***readLine()*** method of the BufferedReader class is used for reading lines of text from the console, the file, or other input streams.

```
import java.io.*;

public class ReadWriteFile {
	public static void main (String arg[])
			throws IOException {
		InputStreamReader inputStreamReader =
				new InputStreamReader(System.in);
		BufferedReader bufferStream =
				new BufferedReader(inputStreamReader);

		String readString;

		do {
			System.out.println("Please enter something:");
			System.out.flush();
			readString = bufferStream.readLine();
			System.out.println("Hello User > This is"
					+ " what you wrote ");
			System.out.println(">" + readString);
		} while ( readString.length() != 0 );
	}

}
```

Figure 7.5: Example showing Reading from the keyboard

Output:

Please enter something:

Hi How are you

Hello User > This is what you wrote

>Hi How are you

Please enter something:

Hello User > This is what you wrote

>

In the above code, the **InputStreamReader** constructor takes an object of the InputStream class System.in as its parameter and constructs an InputStreamReader object called inputReader. Passing the InputStreamReader object as a parameter to the constructor of BufferedReader creates a BufferedReader object. The readLine() method of BufferedReader reads the line from the console. This line is then displayed as a message in the screen. You have to press Enter to terminate the program.

Self-Assessment Questions – 4

8. The _____ and _____ classes are abstract classes that support the reading and writing of Unicode character streams.
9. Unicode is used to represent data such that each character is represented by _____ .
10. The most important subclasses of the Reader and Writer classes are _____ And _____ .

6. RANDOM ACCESS FILES

The term random access means that data can be read from or written to random locations within a file. In all the above mentioned streams, data is read and written as a continuous stream of information. Java provides the RandomAccessFile class to perform I/O operations at specified locations within a file.

The RandomAccessFile class implements the **DataInput** and **DataOutput** interfaces for performing I/O using the primitive data types. The RandomAccessFile class also supports permissions like read and write, and allows files to be accessed in read-only and read-write modes.

Creating a Random Access File

There are two ways to create a random access file - using the pathname as a string or using an object of the File class.

RandomAccessFile (*String pathname, String mode*);

RandomAccessFile (*File name, String mode*);

Example:

```
randomFile = new RandomAccessFile ("iotets.txt","rw");

or

File file1 = new File ("iotest.txt");
RandomAccessFile randomFile = new RandomAccessFile (file1, "rw");
```

The second argument is the mode argument that determines whether you have read-only or read/write (rw) access to the file.

The RandomAccessFile class has several methods that allow random access to the content within the file. In table7.7 these methods have been discussed.

Table 7.7: Methods of RandomAccessFile Class

Method	Explanation
void seek (long pos)	Sets the file pointer to a particular location inside the file.
long getFilePointer ()	Return the current location of the file pointer.
long length ()	Returns the length of the file, in bytes.

7. SERIALIZATION

When you create a file using a text editor, it allows you to store the text to a file. The text stored in the file is the data encapsulated within the file object. This object is permanently available once you store it on the disk. The capability of an object to exist beyond the execution of the program that created is known as **persistence**.

In the above example, the file object continues to exist even after the text editor stops executing. To make the file object persistent, you have to carry out an important step: saving the file. Saving the file object is called **serialization**. **Serialization** is the key to implement persistence. It provides the capability for you to write an object to a stream and read the object at a later stage.

JDK 1.2 provides the **Serializable** interface in the **java.io** package to support object serialization. The process of reading an object and writing it to a file stream is very simple. The **readObject()** method of the **ObjectInputStream** class is used to read objects from the stream.

The Objects that are read should be cast to the appropriate class names. Similarly, the writeObject() method of the ObjectOutputStream class writes objects to a stream.

Self-Assessment Questions – 5

11. Java provides the _____ class to perform I/O operations at specified locations within a file.
12. The RandomAccessFile class implements the _____ and _____ interfaces.
13. The capability of an object to exist beyond the execution of the program that created is known as _____ .
14. _____is the key to implement persistence.

- The Reader and Writer classes support I/O for Unicode character.
- The RandomAccessFile class supports random access operations in files.

8. STREAM API

Java 8 Streams is a new addition to the Java Collections API, which brings a new way to process collections of objects. Thus, streams in the Java Collections API is a different concept than the input and output streams in the Java IO API, even if the idea is similar (a stream of objects from a collection, instead of a stream of bytes or characters). Streams are designed to work with Java lambda expressions.

Lambda Expressions and Functional Interfaces

Lambdas are ways to express computation based on a function abstraction

Syntax: (*params*) -> *expression*

For example, instead of writing a method:

```
void fn (String name) {
     System.out.println(name);
}
```

And invoking this for every 'name' in a collection, we can rewrite this as :

```
name-> System.out.println(name);
```

Functional interfaces are Interfaces with a single abstract method. From Java 8, Java supports methods that can take instances of Functional Interfaces as parameters. Lambda expressions can be used to generate instances of the functional interfaces.

For example, the method map() takes an instance of functional interface named 'Function'. This interface has a single method named 'apply()'.

As a developer, we can invoke the map as follows:

map(name->name+10)

name->name+10 is a lambda expression. name+10 is taken as the implementation for the apply method and this generates a new instance of the implementation of the interface.

Thus functional interface and lambda expressions can be used to define flexible methods in a class.

Generating Streams

With Java 8, Collection interface has two methods to generate a Stream.

- stream() – Returns a sequential stream considering collection as its source.
- parallelStream() – Returns a parallel Stream considering collection as its source.

In Java 8, every class which implements the java.util.Collection interface has a stream method which allows to convert its instances into Stream objects. Therefore, it's trivially easy to convert any list into a stream. Here's an example which converts an ArrayList of Integer objects into a Stream:

```
//Create an Arralist
List<Integer> myList= new ArrayList<Integer>();
myList.add(1);
myList.add(5);
myList.add(8);
//Convert it into a Stream
Stream<Integer> myStream=myList.stream()
```

Once a stream object is created, there are a variety of methods to transform it into another Stream object. We can perform one or more intermediate operation and then one final terminal operation.

"forEach" Method:

Stream has provided a new method 'forEach' to iterate each element of the stream. The following code segment shows how to print 10 random numbers using forEach.

```
Random random=new Random();
random.ints().limit(10).forEach(System.out::println);
```

"map" Method: It takes a lambda expression as its only argument, and uses it to change every individual element in the stream. Its return value is a new Stream object containing the changed elements.

In the following example, map is used to convert all elements in an array of strings to uppercase.

First, the array is converted into a stream.

```
String[] myArray=new string[]{"bob","alice","paul","ellie"};
Stream<String> myStream= Arrays.stream(myArray);
```

Then using map method, passing a lambda expression, string is converted to uppercase.

```
Stream<String> myNewStream= myStream.map(s->s.toUpperCase());
```

The Stream object returned contains the changed strings. To convert it into an array, you use its toArray method:

```
String[] myNewArray=myNewStream.toArray(String[]::new);
```

"filter" Method: The 'filter' method is used to eliminate elements based on a criteria. The

```
List<String>strings=Arrays.asList("abc","","bc", \
                                  "efg","abcd","","jkl");
```

following code segment prints a count of empty strings using filter.

"limit" method: The 'limit' method is used to reduce the size of the stream. The following code segment shows how to print 10 random numbers using limit.

```
Random random=new Random();
random.ints().limit(10).forEach(System.out::println);
```

"sorted" method: The 'sorted' method is used to sort the stream. The following code segment shows how to print 10 random numbers in a sorted order

```
Random random=new Random();
random.ints().limit(10).sorted().forEach(System.out::println);
```

"parallelStream" method: parallelStream is the alternative of stream for parallel processing. Take a look at the following code segment that prints a count of empty strings using parallelStream.

```
List<String>strings=Arrays.asList("abc","",\
                          "bc","efg","abcd","","jkl");
//Get count of empty string
int count=strings.parallelStream().filter( \
                    string->string.isEmpty()).count();
```

"collectors" method: Collectors are used to combine the result of processing on the elements of a stream. Collectors can be used to return a list or a string.

```
List<String> strings = Arrays.asList("abc", "", "bc", "efg",
                           "abcd", "","jkl");
List<String> filtered = string.stream()
               .filter(string -> !string.isEmpty())
               .collect(Collectors.toList());

System.out.println("Filtered List:" + filtered);
String mergedString = strings.stream()
               .filter(string -> !string.isEmpty())
               .collect(Collectors.joining(","));
System.out.println("Merged String:" + mergedString);
```

9. SUMMARY

- The classes of the java.io package provide support for input and output operations in Java programs.
- A stream is a sequence of bytes traveling from a source to a destination over a communication path.
- InputStream and OutputStream are superclasses used for reading from and writing to byte streams.
- The Reader and Writer classes support I/O for Unicode character.
- The RandomAccessFile class supports random access operations in files.

10. TERMINAL QUESTIONS

1. What are the uses of a stream class?
2. What is the syntax of FileInputStream and FileOutputStream ?
3. What are the different methods under DataInputStream and DataOutputStream?
4. What are the uses of random access file over sequential access file?

11. ANSWERS

Self-Assessment Questions

1. Streams.
2. Input, Output.
3. Filters.
4. Markable Streams.
5. void write(int n).
6. FileInputStream, FileOutputStream.
7. DataInputStream, DataOutputStream.
8. Reader, Writer.
9. 16 bits.
10. InputStreamReader, OutputStreamWriter.
11. RandomAccessFile.
12. DataInput, DataOutput.
13. Persistence.

14. Serialization.

Terminal Questions

1. Streams are powerful because they abstract the details of input/output (I/O) operations. (Refer Section 1)
2. FileInputStream inputfile = new FileInputStream ("Employee.dat"); FileOutputStream outputfile = new FileOutputStream ("binus.dat"); (Refer Section 3)
3. Methods of the DataInputStream class

Method	Explanation
boolean readBoolean ()	Reads one byte and returns true if that byte isnonzero, false if it is zero.
byte readByte ()	Reads a byte as an 8-bit signed value.
char readChar ()	Reads a Unicode character.
int readInt ()	Reads an integer value.

Methods of the DataOutputStream Class

Method	Explanation
void writeChar (int v)	Writes a character to the output stream.
void writeLong (long v)	Writes a long integer to the output stream.
void writeInt (int v)	Writes an integer to the output stream.

(Refer section 3)

4. The term random access means that data can be read from or written to random locations within a file. (Refer Section 5)

12. REFERENCES

- Schildt, Herbert. Java: The Complete Reference. New York: McGraw-Hill Education, 2018.