Third Year Diploma Courses in Computer Science & Engineering, Computer Engineering, Computer Technology and Information Technology Branch.

Java Programming

As per MSBTE 'I' Scheme Syllabus
JPR-22412

Unit- VI
Managing Input Output Files in java

Total Marks- 08

.

Prof. Gunwant V. Mankar
B.E(IT), M.Tech(CSE), AMIE, MIAEng, MSCI
HOD *CSE Dept***.**
(BTC- School of Diploma in Engineering, Ballarpur)
e-mail:- info@gunwantmankar.com
Website:- www.gunwantmankar.com)

Unit- VI- Managing Input Output Files in Java

6.1. Stream Classes

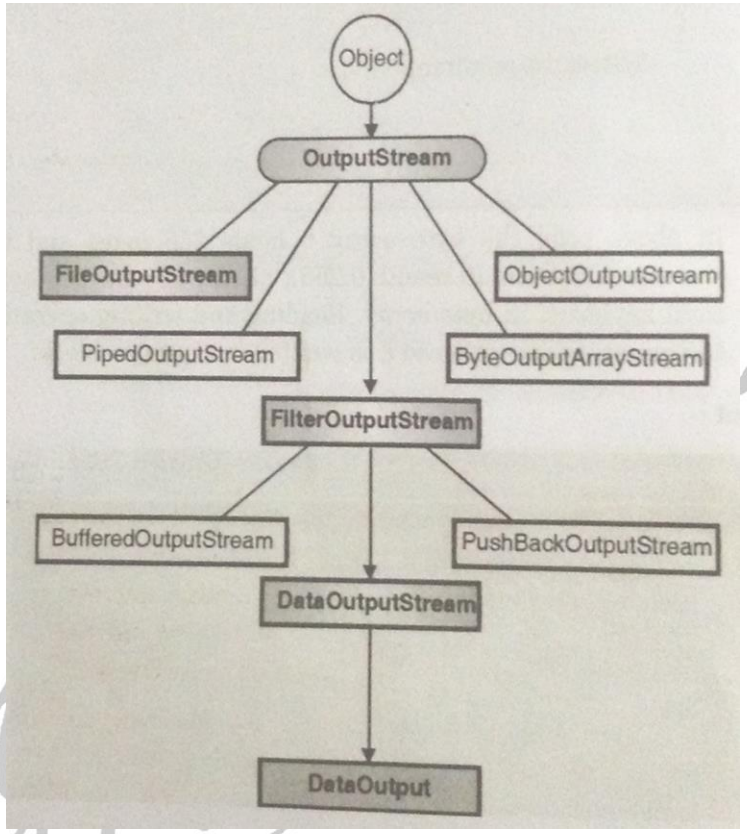

Fig. Hierarchy of java stream classes

1. **What are stream classes ? List any two input stream classes from character stream [S-15, S-16]**

Definition:

The java. IO package contain a large number of stream classes that provide capabilities for processing all types of data. These classes may be categorized into two groups based on the data type on which they operate.

1. Byte stream classes that provide support for handling I/O operations on bytes.
2. Character stream classes that provide support for managing I/O operations on characters.

Character Stream Class can be used to read and write 16-bit Unicode characters. There are two kinds of character stream classes, namely, reader stream classes and writer stream classes

Reader stream classes:-

It is used to read characters from files. These classes are functionally similar to the input stream classes, except input streams use bytes as their fundamental unit of information while reader streams use characters

Input Stream Classes

1. BufferedReader
2. CharArrayReader
3. InputStreamReader
4. FileReader
5. PushbackReader
6. FilterReader
7. PipeReader
8. StringReader

2. What are streams ? Write any two methods of character stream classes. [W-15]

Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information (i.e it takes the input or gives the output). A stream is linked to a physical device by the Java I/O system.

All streams behave in the same manner, even if the actual physical devices to which they are linked differ. Thus, the same I/O classes and methods can be applied to any type of device.

Java 2 defines two types of streams: byte and character.

Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data.

Character streams provide a convenient means for handling input and output of characters.

They use Unicode and, therefore, can be internationalized. Also, in some cases, character streams are more efficient than byte streams.

The Character Stream Classes

Character streams are defined by using two class hierarchies. At the top are two abstract classes, **Reader** and **Writer**. These abstract classes handle Unicode character streams. Java has several concrete subclasses of each of these.

Methods of Reader Class

1) void mark(int *numChars*) : Places a mark at the current point in the input stream that will remain valid until numChars characters are read.

2) boolean markSupported( ) : Returns **true** if **mark( )** / **reset( )** are supported on this stream.

3) int read( ) : Returns an integer representation of the next available character from the invoking input stream. –1 is returned when the end of the file is encountered.

4) int read(char *buffer*[ ]) : Attempts to read up to buffer. Length characters into buffer and returns the actual number of characters that were successfully read. –1 is returned when the end of the file is encountered.

5) abstract int read(char *buffer*[ ],int *offset*,int *numChars*): Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read.–1 is returned when the end of the file is encountered.

6) boolean ready( ): Returns **true** if the next input request will not wait. Otherwise, it returns **false**.

7) void reset( ): Resets the input pointer to the previously set mark.

8) long skip(long *numChars*) :- Skips over *numChars* characters of input, returning the number of characters actually skipped.

9) abstract void close( ) :- Closes the input source. Further read attempts will generate an IOException

Writer Class

Writer is an abstract class that defines streaming character output. All of the methods in this class return a **void** value and throw an **IOException** in the case of error

Methods of Writer class are listed below: -

1) abstract void close( ) : Closes the output stream. Further write attempts will generate an **IOException**.

2) abstract void flush( ) : Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.

3) void write(int*ch*): Writes a single character to the invoking output stream. Note that the parameter is an **int**, which allows you to call **write** with expressions without having to cast them back to **char**.

4) void write(char *buffer*[ ]): Writes a complete array of characters to the invoking output stream

5) abstract void write(char *buffer*[ ],int *offset,* int *numChars*) :- Writes a subrange of *numChars* characters from the array *buffer*, beginning at *buffer*[*offset*] to the invoking output stream.

6) void write(String *str*): Writes *str* to the invoking output stream.

7) void write(String *str,* int *offset*,int *numChars*): Writes a sub range of *numChars* characters from the array *str,* beginning at the specified *offset*.

[Note: any two methods from above list to be considered]**

3. **What is use of stream classes ? Write any two methods FileReader class.[W-14]**

- An I/O Stream represents an input source or an output destination.
- A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.
- Some streams simply pass on data; others manipulate and transform the data in useful ways. Java‟s stream based I/O is built upon four abstract classes: InputStream, OutputStream, Reader, Writer.
- They are used to create several concrete stream subclasses, the top level classes define the basic functionality common to all stream classes.
- InputStream and OutputStream are designed for byte streams and used to work with bytes or other binary objects.
- Reader and Writer are designed for character streams and used to work with

character or string.

1. public int read()throws IOException - Reads a single character.

2. public int read(char[ ] cbuf, int offset, int length) throws IOException - Reads characters into a portion of an array.

3. public void close()throws IOException - Closes the stream and releases any system resources associated with it. Once the stream has been closed, further read(), ready(), mark(), reset(), or skip() invocations will throw an IOException. Closing a previously closed stream has no effect

4. public boolean ready()throws IOException - Tells whether this stream is ready to be read. An InputStreamReader is ready if its input buffer is not empty, or if bytes are available to be read from the underlying byte stream

5. public void mark(int readAheadLimit) throws IOException -Marks the present position in the stream. Subsequent calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support the mark() operation.

6. public void reset()throws IOException - Resets the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream, for example by repositioning it to its starting point. Not all character-input streams support the reset() operation, and some support reset() without supporting mark().

4. **Write any two methods of File and FileInputStream class each.[S-15, W-15]**

File Class Methods

1. **String getName( )** - returns the name of the file.

2. **String getParent( )** - returns the name of the parent directory.

3. **boolean exists( )** - returns true if the file exists, false if it does not.

4. **void deleteOnExit( )** -Removes the file associated with the invoking object when the Java Virtual Machine terminates.

5. **boolean isHidden( )**-Returns true if the invoking file is hidden. Returns false otherwise.

FileInputStream Class Methods:

1. **int available( )**- Returns the number of bytes of input currently available for reading.
2. **void close( )**- Closes the input source. Further read attempts will generate an IOException.
3. **void mark(int numBytes)** -Places a mark at the current point in the inputstream that will remain valid until numBytes bytes are read.
4. **boolean markSupported( )** -Returns true if mark( )/reset( ) are supported by the invoking stream.
5. **int read( )**- Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
6. **int read(byte buffer[ ])**- Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.

5. Explain serialization in relation with stream class. [W-14,W-15, S-16]

Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

Serialization is also needed to implement Remote Method Invocation (RMI). RMI allows a Java object on one machine to invoke a method of a Java object on a different machine. An object may be supplied as an argument to that remote method. The sending machine serializes the object and transmits it. The receiving machine deserializes it.

Example:

Assume that an object to be serialized has references to other objects, which, inturn, have references to still more objects. This set of objects and the relationships among them form a directed graph. There may also be circular references within this object graph. That is, object X may contain a reference to object Y, and object Y may contain a reference back to object X. Objects may also contain references to themselves. The object serialization and deserialization facilities have been designed to work correctly in these scenarios. If you attempt to serialize an object at the top of an object graph, all of the other referenced

objects are recursively located and serialized. Similarly, during the process of deserialization, all of these objects and their references are correctly restored.

6. Write a program to copy contents of one file to another file using character stream class.[S-15]

```
import java.io.*;
class CopyData
{
        public static void main(String args[ ])
        {

        //Declare input and output file stream

        FileInputStream  fis= null;   //input stream

        FileOutputStream  fos=null;  //output Stream
        //Declare a variable to hold a byte

        byte  byteRead;
         try
        {
        // connect fis to in.dat
        fis=new FileInputSream("in.dat");
        // connect fos to out.dat
        fos= new FileOutputStream("out.dat");
        //reading bytes from in.dat and write to out.dat

        do
        {
        byteRead =(byte)fis.read( );
        fos.write(byteRead);
        }
        while(byteRead != -1);
        }

        Catch(FileNotFoundException e)

        {

        System.out.println("file not found");

        }

        Catch(IOException  e)

        {

        System.out.pritln(e.getMessage( ));

        }

        finally     // close file
```

```
        {
         try
        {
        fis.close( );
        fos.close( );
        }
        Catch(IOException  e)
        {  }
        }
        }
}
```

7. **What is use of ArrayList Class ? State any three methods with their use from ArrayList.[W-15, S-16]**

Use of ArrayList class:

1. ArrayListsupports dynamic arrays that can grow as needed.
2. ArrayListis a variable-length array of object references. That is, an ArrayListcan dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

Methods of ArrayList class :

1. void add(int index, Object element) Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >size()).

2. boolean add(Object o) Appends the specified element to the end of this list.

3. booleanaddAll(Collection c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.

4. booleanaddAll(int index, Collection c) Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.

5. void clear() Removes all of the elements from this list.

6. Object clone() Returns a shallow copy of this ArrayList.

7. boolean contains(Object o) Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).

8. void ensureCapacity(intminCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

9. Object get(int index) Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

10. intindexOf(Object o) Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

11. intlastIndexOf(Object o) Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

12. Object remove(int index) Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size()).

13. protected void removeRange(intfromIndex, inttoIndex) Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

14. Object set(int index, Object element) Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

15. int size() Returns the number of elements in this list.

16. Object[ ] toArray() Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.

17. Object [] toArray(Object[] a) Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

18. void trimToSize() Trims the capacity of this ArrayList instance to be the list's current size.

8. Write syntax and function of following methods of Date class :
i) getTime ( ) ii) getDate ( ) [S-15]

The **Date** class encapsulates the current date and time.
i. getTime():
Syntax:
long getTime( )
Returns the number of milliseconds that have elapsed since January 1, 1970.

ii. getDate()
Syntax:
public int getDate()
Returns the day of the month. This method assigns days with the values of
1 to 31.

9. Write syntax and function of following methods of date class :
1) sethme ( ) 2) getDay ( ) [W-14]

1. setTime():

void setTime(long time):
the parameter time - the number of milliseconds.
Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT

2.getDay()

int getDay():
Returns the day of the week represented by this date.
The returned value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

10. Write any four mathematical functions used in Java.[W-14]

1) min() :
Syntax: static int min(int a, int b)
Use: This method returns the smaller of two int values.

2) max() :

Syntax: static int max(int a, int b)
Use: This method returns the greater of two int values.

3) sqrt()
Syntax: static double sqrt(double a)
Use : This method returns the correctly rounded positive square root of a double
value.

4) **pow()** :
Syntax: static double pow(double a, double b)
Use : This method returns the value of the first argument raised to the power of the
second argument.

5) exp()
Syntax: static double exp(double a)
Use : This method returns Euler's number e raised to the power of a double value.

6) round() :
Syntax: static int round(float a)

Use : This method returns the closest int to the argument.

7) abs()
Syntax: static int abs(int a)
Use : This method returns the absolute value of an int value.

11. What is use of setclass ? Write a program using setclass.[W-14]

The Set interface defines a set. It extends Collection and declares the behavior of a collection that does not allow duplicate elements. Therefore, the add( ) method returns false if an attempt is made to add duplicate elements to a set.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

The methods declared by Set are summarized in the following table

Sr.No	Methods	Description
1	add()	Adds an object to the collection
2	clear()	Removes all objects from the collection
3	contains()	Returns true if a specified object is an element within the collection

4	isEmpty()	Returns true if the collection has no elements
5	iterator()	Returns an Iterator object for the collection which may be used to retrieve an object
6	remove()	Removes a specified object from the collection
7	size()	Returns the number of elements in the collection

Following is the example to explain Set functionality:

```
import java.util.*;
public class SetDemo
        {
        public static void main(String args[])
        {
        int count[] = {34, 22,10,60,30,22};
        Set<Integer> set = new HashSet<Integer>();
        try{
        for(int i = 0; i<5; i++){
        set.add(count[i]);
        }
        System.out.println(set);
        TreeSet sortedSet = new TreeSet<Integer>(set);
        System.out.println("The sorted list is:");
        System.out.println(sortedSet);
System.out.println("The First element of the set is: "+
(Integer)sortedSet.first());

System.out.println("The last element of the set is: "+ (Integer)sortedSet.last());
}
catch(Exception e){}
}
}
```

Executing the program.

```
[34, 22, 10, 30, 60]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

12. State syntax and describe any two methods of map class.[S-16]

The Map Classes Several classes provide implementations of the map interfaces. A map is an object that stores associations between keys and values, or key/value pairs. Given a key, you can find its value. Both keys and values are objects. The keys must be unique, but the values may be duplicated. Some maps can accept a null key and null values, others cannot.

Methods:

void clear // removes all of the mapping from map

booleancontainsKey(Object key) //Returns true if this map contains a mapping for the specified key.

Boolean conainsValue(Object value)// Returns true if this map maps one or more keys to the specified value

Boolean equals(Object o) //Compares the specified object with this map for equality.

END OF UNIT.