Chapter 4 (Java IO)

(Note: This document is dedicated for BE Computer 7th Sem. or BCA 5th Sem., so it is assumed that the already have knowledge what is IO actually. Further discussion on the example will be in class)

Introduction

While working with program we may have to deal with external IO operation like reading files, writing files, socket programming etc. Java IO deals with all IO operations.

Data Sinks

- Resource for Data
- Types of Data Sinks
  - File
    - Read/write from file
  - Memory
    - Read/write from memory
  - Pipe
    - Read/write across threads
- Types of Data
  - Character
    - Input from Keyboard and output to screen
    - As memory sinks and pipe sinks
  - Raw data
    - Network/sockets data or data read from other programming language.
  - Formatted Data
    - Formatted object, images or video files
  - Compressed Data
    - Gzip, jar etc

Character Sink Classes

Type	Classes
File	FileReader FileWriter
Memory	CharArrayReader CharArrayWriter StringReader StringWriter
Pipe	PipedReader PipeWriter

Data Sink Classes

Type	Classes
File	FileInputStream FileOutputStream
Memory	ByteArrayInputStream ByteArrayOutputStream
Pipe	PipedInputStream PipeOutputStream

Introduction to Streams

- Java uses the concept of streams to represent the ordered sequence of data, a common characteristic shared by all the input/output devices as started above.
- For example, if we are using terminal in one side, the other side should not always be terminal, it may be file or socked output.
- Standard Input, File Stream, Standard Output, File System etc all are handled through streams.
- Streams are used to access resources
- Different classes handle different types of data
- Streams are often stacked to process data
- Exception must be handled
- I/O classes were redesigned in Java 1.1 (older classes still exist
- It contains binary information
- The streams are basically divided into InputStream and OutputStream.
- Streams can also be classified as Data(Byte) Streams and Character Streams
- Again both data stream and character streams are divided into Input Stream and Output Stream
- It don't care underlying source/destination
- **Data and Character Streams**
    - OutputStream
        - Write program data out to stream.
    - InputStream
        - Read data from data sinks into a program.
    - Writer
        - Work with character data to write into stream
    - Reader
        - Work with character data to read from stream
    - InputStreamReader
        - Allow to read in data and character into program from stream.
    - OutputStreamWriter.
        - Allow to read in data and character into program from stream.

I/O Exception Class Hierarchy

- IOException
- FileNotFoundException
- InterruptedIOException
- ObjectStreamException

I/O Classes

- InputStrem (abstrace class/base class)
    - void close()
    - int read()
    - int read(byte[] b)
    - int read(byte[] b, int off, int len)
    - void reset()
    - long skip(long n)
- OutputStream (abstrace class/base class)
    - void close()
    - void flush()
    - void write(int b)
    - void write(byte[] b)
    - void write(byte[] b, int off, int len)

File

- Most of the java.io operates on stream, but file class doesn't.
- It deals directly with files and the file system.
- The File class does not specify how information is retrieved from or stored in files; it describes the properties of a file itself.
- A File object is used to obtain or manipulate the information associated with a disk file, such as the permissions, time, date, and directory path, and to navigate subdirectory hierarchies.
- There are several restriction to use file in Applet
- A directory is treated as simply a file with one property added.
- Following constructors can be used to crate File objects:
  File(String directoryPath)
  File(String directoryPath,String Filename)
  File(File dirObj,String filename)
  File(URI uriObj)
- Here, directoryPath is the pathname
- filename is the name of the file
- dirObj is a File object that specifies a directory
- uriObj is a URI object that describes a file.
- Example
  File f1=new File("/");
  File f2=new File("/","test.dat");
  File f3=new File(f1,"test.dat");
- File defines many methods that obtain the standard properties of a file object.
- Methods related to File object

getName( )	getPath( )	getAbsolutePath( )
getParent( )	exists( )	canWrite( )
canRead( )	isDirectory( )	isFile( )
isAbsolute( )	lastModified( )	length( )
boolean renameTo( )	void deleteOnExit( )	isHidden( )
boolean delete( )	Boolean setLastModified(long millisec)	
Boolean setReadOnly( )		

Example

```
//fileproperties.java
import java.io.*;

public class fileproperties
{
	static void property(String s)
	{
		System.out.println(s);
	}
	public static void main(String arg[])
	{
		File f=new File(".\\a\\L7Q1.java");
		property("File Name: " +f.getName());
		property("File Path: " +f.getPath());
		property("Absulate path: " +f.getAbsolutePath());
		property("Parent : " +f.getParent());
		property("Existence: " +f.exists());
		property("Is File: " +f.isFile());
		property("Is Directory: " +f.isDirectory());
		property("Read/Write: " +f.canWrite());
		property("Read only: " +f.canRead());
```

```
                property("Is absolute: " +f.isAbsolute());
                property("Is Hidden: " +f.isHidden());
                property("File Size: " +f.length() +" Bytes");
                property("Is Modified on : " +f.lastModified());
        }
}
```

Directories

- A directory is a File that contains a list of other files and directories.
- When we create a File object and it is a directory, the **isDirectory( )** method will return true.
- list( ) method is used to extract the list of other files and directories inside
- It has two forms

  String [ ] list ( )

Example

```
//dir.java
import java.io.*;

public class dir
{

        public static void main(String arg[])
        {
                File f=new File("..\\iolab.txt"); // iolab.txt is in parent directory
                if (f.isDirectory())
                {
                        System.out.println("The content of directory lab7 is : ");
                        String s[]=f.list();
                        for(int i=0;i<s.length;i++)
                                System.out.println(s[i]);
                }
        }
}
```

Byte Stream Classes through Reading from and Writing to a File

- Byte stream classes are used to provide functional future for creating and manipulating streams and files for reading and writing bytes.
- Java provides both input and output byte stream classes.
- The FileInputStream class allows you to read from a file in the form of a stream.
- A stream is a path along which data flows.
- We can create an object of the class using a File class.
- The FileOutputStream class allows us to write the output to a file steam.

Example

```
//CopyFile.java
//Copy one file to another
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyFile
{
   public static void main(String[] args)
   {
      if(args.length !=2){
         System.out.println("Usage: java CopyFile <infile> <outfile>");
```

```
            return;
         }
         try{
            int value;
            FileInputStream in = new FileInputStream(args[0]);
            FileOutputStream out = new FileOutputStream(args[1]);
            while((value=in.read()) != -1)
               out.write(value);
            in.close();
            }catch(IOException e)
            {
               System.out.println(e);
            }
      }
   }
```

Character Stream classes through Reading from and Writing to a File

- The character stream classes supports Input and Output for Unicode characters.
- Character Stream classes namely reader and writer stream class provide read and write 16 bit Unicode characters.
- FileReader class is used to read characters from files.
- The FileWriter class provides writing characters to a file.
- These classes are very similar to the Byte stream Input and Output classes; the only difference is fundamental unit of information (Byte/Character).

Example

```
//Enter your name in console and get welcome message
import java.io.*;

public class BasicIO
{
	public static void main(String[] args)
	{
		try
		{
			InputStreamReader isr = new InputStreamReader( System.in );
			BufferedReader stdInput = new BufferedReader (isr);

			System.out.print("Print enter your Name:");
			String inputData = stdInput.readLine();
			System.out.println("Welcome " + inputData);
		}catch(IOException ioe)
		{
			System.out.println("An I/O Error has occoured");
		}
	}
}
```

Example 2

```
//findCube.java
//Find the square of a number inputted by the user.
	public class findCube
	{
		public static void main(String args[]) throws IOException
		{
			String str;
```

```
                int num;
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter the Number :");
                str=br.readLine();
                System.out.print("\n");
                num=Integer.parseInt(str);
                age=age*age;
                System.out.println("Square of the Number is: "+num);
        }
}
```

Random Access File

- The RandomAccessFile enables us to read and write bytes, text and java data types to any location in a file.
- **Java.io.RandomAccessFile class** provides the ability to read and write data from or write any specified location in a file.
-
- This class implements DataInput and DataOutput interfaces.
- The random access means that data can be read from or written to random location within a file.
- This class also provides permissions like read and write, and allows files to be accessed in read only and read write mode.
- There are two ways to create a random access file.
    - Using an object of the file class
    - Using the path name as a String.

  RandomAccessFile(File name,String Mode)
  RandomAccessFile(String pathname,String Mode)
- We can use one of the following two mode Strings
    - "r" for reading only
    - "rw" of read and write.
- The following program creates a random access file and opens it in the form of "rw" mode.
- We can read and write the primitive data types through readXXX( ) and WriteXXX( ) methods.
- The RandomAccessFile Class has several methods that allow random access to the content within file.
- The **void seek(long position)**, it sets the file pointer to a particular location inside the file.
- The **length( )** method returns the length of the file in terms of bytes.
- **Long getFilePointer( )** returns the current byte location of the file pointer
- **int skipBytes(n)** jumps the pointer by n bytes from its current location.

Example

```
//randomfile.java
import java.io.*;
class randomfile
{
        public static void main(String args[ ]) throws IOException
        {
                try
                {
                        RandomAccessFile r1=new RandomAccessFile("in.dat","rw");
                        r1.writeShort(10);
                        r1.writeInt(20000);
```

```
                        r1.writeDouble(342.234);
                        r1.writeChar('a');
                        r1.writeBoolean(true);
                        r1.writeLong(545234325);
                        r1.writeFloat((float)42342.343);
                        r1.seek(0);
                        System.out.println(r1.readShort());
                        System.out.println(r1.readInt());
                        System.out.println(r1.readDouble());
                        System.out.println(r1.readChar());
                        System.out.println(r1.readBoolean());
                        System.out.println(r1.readLong());
                        System.out.println(r1.readFloat());
                        r1.close();
                }
                catch(FileNotFoundException e)
                {
                        System.out.println(e.getMessage());
                }
        }
}
```

Sequence Input Stream

- The SequenceInputStream class allows you to concatenate multiple InputStreams.
- The SequenceInputStream use the following constructor:
  - SequenceInputStream(InputStream is1,InputStream is2)
- It read requests from the is1 until it runs out and then switches over the is2.
- (try it your self)

Serialization

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

```
import java.io.*;

public class SerializationEx
{
        public static void main(String arg[])
        {
                try
                {
                        X obj1=new X("hi",-7,2.0);
                        X obj2;
                        System.out.println("obj1"+obj1);
                        FileOutputStream fos=new FileOutputStream("serial");
                        ObjectOutputStream oos=new ObjectOutputStream(fos);
                        oos.writeObject(obj1);
                        oos.flush();
                        oos.close();
                        FileInputStream fis=new FileInputStream("serial");
                        ObjectInputStream ois=new ObjectInputStream(fis);
                        obj2=(X)ois.readObject();
                        ois.close();
```

```
                System.out.println("obj2"+obj2);
            }
            catch(Exception e)
            {
                System.out.println("the error is : "+e);
            }
        }
}

class X implements Serializable
{
        String s;
        int i;
        double d;

        public X(String s,int i,double d)
        {
            this.s=s;
            this.i=i;
            this.d=d;
        }
        public String toString()
        {
            return "s="+s+";i="+i+";d="+d;
        }
}
```