CHAPTER - 9

FILES

Data storage in variables and arrays is temporary. The data is loss when the program terminates. Files are used for long term storage of large amounts of data, even after the program that created the data terminates.

9. File

A File is a collection of related records placed in a specific area on the disk. Data that is stored in files is often called persistent data. A record is a collection of several fields and the field is a collection of characters. Each character in a computer character set is represented as a pattern of 1's and 0's.

Creation, Updating, Managing data etc., using file is known as 'File Processing'.

9.1 *File Class*

java.io package supports for Input & Output operations on files. This package provides a class known as File Class to creating Files. File Object is used to manipulate or obtain the information associated with a disk file, such as permission, data, directory path and soon.

The following constructors are used to create a File Object.

File (String directorypath)

File (String directorypath, String filename)

File (File object, String filename)

Methods:

1. boolean canRead( )
2. boolean canWrite( )

3. boolean exists( )
4. String getName( )
5. String getParent( )
6. String getPath( )
7. boolean isDirectory( )
8. boolean isFile( )
9. long length( )
10. long lastModified( )
11. boolean isHidden( )
12. String getAbsolutePath( )
13. boolean isAbsolute( )
14. String [] list( )

Example:

```
import java.io.*;
class FileDemo
{
    public static void main(String[] args)
    {
File f=new File("D:/shabbir/wish.java");
System.out.println(f.canRead());                    //true
System.out.println(f.canWrite());                   //true
System.out.println(f.exists());                     //true
System.out.println(f.getName());                    //wish.java
System.out.println(f.getParent());                  //D:\shabbir
System.out.println(f.getPath());                    //D:\shabbir\wish.java
System.out.println(f.isDirectory());                //false
System.out.println(f.isFile());                     //true
System.out.println(f.length());                     //210
System.out.println(f.lastModified());               //1165345688000
System.out.println(f.isHidden());                   //false
System.out.println(f.getAbsolutePath());            //D:\shabbir\wish.java
System.out.println(f.isAbsolute());                 //true
    }
}
```


Example: Write a program to display all files in the given directory

```
import java.io.*;
class All
{
    public static void main(String[] args)
    {
    File f=new File("D:/shabbir");
    String str[]=null;
    if(f.isDirectory())
    str=f.list();
    for(int i=0;i<str.length;i++)
    System.out.println(str[i]);
    }
}
```

9.2 Filename Filter Interface

list( ) method, it returns all the files existed in the given directory. When we want to limit the number of files returned by the list( ) method to include only those files that match a certain filename pattern or filtered, such case use the second form of list( ) method as

Syntax: String [] list(FilenameFilter obj)

Here, obj is an object of class that implements the FilenameFilter interface. FilenameFilter interface defines only a single method, accept( ), which is called once for each file in a list. Its general format is as

Syntax: boolean accept(File directory, String filename)

The accept( ) method returns true for files in the directory specified by directory that should be included in the list; otherwise false.

Example: Write a program to list the filenames that are ended with .java extension

```
import java.io.*;
class Filter implements FilenameFilter
```

```
{
    public boolean accept(File f,String str)
    {
     if(str.endsWith(".java"))
     return true;
     else
     return false;
    }
}
class AllDemo
{
    public static void main(String[] args)
    {
    File f=new File("D:/shabbir");
    Filter obj=new Filter();
    String str[]=null;
    if(f.isDirectory())
    str=f.list(obj);
    for(int i=0;i<str.length;i++)
    System.out.println(str[i]);
    }
}
```

9.3 Streams

A Stream is an abstraction that either produces of consuming information i.e., flow of data. A stream is linked to a physical device by the java I/O system. So, that an input stream can abstract many kinds of input from a disk file, keyboard, network socket etc., output stream refers to file, network connection, printer etc.,

When a file is opened, an object is created and a stream is associated with the object. Three stream objects are created by the java system automatically when the program execution begins. They are

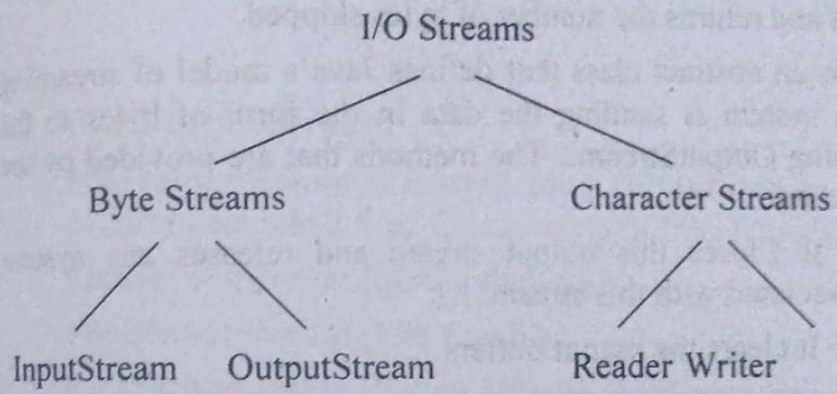
1) System.in 2) System.out 3) System.err

System.in: It is a standard input stream object. It enables a program to input bytes from the keyboard.

System.out: It is a standard output stream object. It enables a program to output data to the screen.

System.err: It is a standard error stream object. It enables a program to output error message to the screen.

9.4 I/O Stream Hierarchy



Java 2 defines two types of Streams called Byte Streams and Character Streams.

Byte Stream provides a convenient way for handling input and output of bytes of data.

Character Stream provides a convenient way for handling input and output of characters of data.

Byte Streams: Byte Stream classes are used for processing the data in the form of bytes. This class defines two important abstract classes called InputStream and OutputStream.

InputStream: It is an abstract class that defines Java's model of streaming byte input. The system is reading the data in the form of bytes from the physical device using InputStream. The methods that are provided by the InputStream class are

a) **int available( ):** Returns number of bytes of input currently available for reading

b) **void close( ):** Closes this input stream and releases any system resources associated with the stream.

c) **int read( ):** Reads the next byte of data from the input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.

d) **int read(byte []b):** It reads upto b.length bytes into b and return the actual number of bytes that were successfully read. –1 is returned when the end of file is encountered.

e) **int read(byte []b, int off, int n):** It attempt to read n number of bytes into byte array b starting from the off and returns number of bytes when successfully. –1 is returned when the end of file is encountered.

f) **long skip(long n):** Skips over and discards n bytes of data from this input stream and returns the number of bytes skipped.

OutputStream: It is an abstract class that defines Java's model of streaming byte output. The system is sending the data in the form of bytes to the physical device using OutputStream. The methods that are provided by the OutputStream class are

a) **void close( ):** Closes this output stream and releases any system resources associated with this stream.

b) **void *flush( ):*** It clears the output buffers.

c) **void write(int b):** Write a single byte b to an output stream.

d) **void write(byte buf[]):**Writes buf.length bytes from the specified byte array to this output stream.

e) **void write(byte buf[], int off, int len):** Writes len bytes from the specified byte array starting at offset off to this output stream.

9.5 File Input Stream

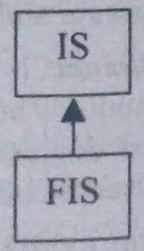
File Input Stream is derived from **Input Stream** and used to read bytes of data from a file. Constructors of FileInputStream are

File Input Stream (String filepath)

File Input Stream (File fileobj)

Here, filepath is the full path name of the file

fileobj is a File object that describes the file.



Example:

```
import java.io.*;
class FISDemo
{
    public static void main(String[] args)throws IOException
    {
    FileInputStream fis=new FileInputStream("d:/shabbir/wi.txt");
    System.out.println(fis.available());
    int k;
    while((k=fis.read())!=-1)
    System.out.print((char)k);
    fis.close();
    }
}
```


9.6 FileOutputStream

FileOutputStream is derived from **OutputStream** and used to send bytes of data into a file. Constructors of FileOutputStream are

FileOutputStream (String filepath)

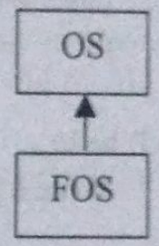
FileOutputStream (File fileobj)

FileOutputStream (String filepath, boolean append)

Here, filepath is the full path name of the file

fileobj is a File object that describes the file

if append is true, then the file is in append mode.



Example 1:

```
import java.io.*;
class FOSDemo1
{
    public static void main(String[] args)throws IOException
    {
     FileOutputStream fos=new FileOutputStream("d:/shabbir/tt.txt");
     fos.write(65);              //O/P:  A
     fos.close();                FileOutputStream                    fos1=new
     FileOutputStream("d:/shabbir/tt.txt",true);
     byte buf[]={66,67,68,69,70};
     fos1.write(buf);            //O/P:  ABCDEF
     fos1.close();
    }
}
```

Example 2:

```
import java.io.*;
class FO SDemo2
{
      public static void main(String[] args)throws IOException
      {
       FileOutputStreamfos=new FileOutputStream("d:/shabbir/wi1.txt");
       String s=new String("hello");
       byte b[]=s.getBytes();  //this method converts string into byte array
       fos.write(b);
       fos.close();
      }
}
```

//O/P:hello

Write a program to copy the contents of a file into another using FileInputStream and FileOutputStream.

```
import java.io.*;
class CopyDemo
{
      public static void main(String[] args)throws IOException
      {
       FileInputStream fis=new FileInputStream("d:/shabbir/wi.txt");
       FileOutputStream fos=new FileOutputStream("d:/shabbir/ci.txt");
       int k;
       while((k=fis.read())!=-1)
       fos.write(k);
       fos.close();
       fis.close();
      }
}
```

9.7 Character Streams

Character Stream classes are used for processing the data in the form of characters. This class defines two important abstract classes called Reader and Writer.

***Reader*:** It is an abstract class that defines Java's model of streaming character input. The system is reading the data in the form of characters from the physical device using Reader. The methods that are provided by the Reader class are

a) **void close( ):** Closes the input source. Further read attempts will generate an IOException

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

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

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

e) **long skip(long numchars):** Skips over numchars characters of input, returning the number of characters actually skipped.

***Writer*:** It is an abstract class that defines Java's model of streaming character output. The system is sending the data in the form of characters to the physical device using Writer. The methods that are provided by the Writer class are

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

b) **void write(int ch):** Writes a single character to the invoking output stream.

c) **void write(char buf[ ]):** Writes a complete array of characters

9.8 Examples on Thread

EXAMPLE :: 1

```
// Controlling the main Thread.
class CurrentThreadDemo1 {
 public static void main(String args[]) {
  Thread t = Thread.currentThread();
```

```
System.out.println("Current thread displayed as: " + t);

// change the name of the thread
t.setName("My Thread");
System.out.println("After name change displayed as: " + t);

try {
  for(int n = 5; n > 0; n--) {
    System.out.println(n);
    Thread.sleep(1000);
  }
 } catch (InterruptedException e) {
  System.out.println("Main thread interrupted");
 }
}
}
```

OUTPUT:

```
C:\>javac CurrentThreadDemo1.java
C:\>java CurrentThreadDemo1
Current thread displayed as: Thread[main,5,main]
After name change displayed as:Thread[My Thread,5,main]
5
4
3
2
1
```

EXAMPLE :: 2

```
// An example of deadlock.
class Deadlock {
 synchronized void foo(B b) {
  String s= Thread.currentThread().getName();

  System.out.println(s + " entered A.foo");
```



```
  try {
   Thread.sleep(1000);
  } catch(Exception e) {
   System.out.println("A Interrupted");
  }

  System.out.println(s + " trying to call B.last()");
  b.last();
 }

 synchronized void last() {
  System.out.println("Inside A.last");
 }
}

class B {
 synchronized void bar(A a) {
  String s = Thread.currentThread().getName();
  System.out.println(s + " entered B.bar");

  try {
   Thread.sleep(1000);
  } catch(Exception e) {
   System.out.println("B Interrupted");
  }
```

EXAMPLE :: 3

```
// Using join() to wait for threads to finish.

class NewTrd implements Runnable {
 String s; // name of thread
 Thread t;

 NewTrd(String threadname) {
  s = threadname;
```

```
    t = new Thread(this, s);
    System.out.println("New thread: " + t);
    t.start(); // Start the thread
  }

  // This is the entry point for thread.
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println(s + ": " + i);
        Thread.sleep(1000);
      }
    } catch (InterruptedException e) {
      System.out.println(s + " interrupted.");
    }
    System.out.println(s + " exiting.");
  }
}

class DemoJoin {
  public static void main(String args[]) {
    NewTrd o1 = new NewTrd("One");
    NewTrd o2 = new NewTrd("Two");
    NewTrd o3 = new NewTrd("Three");

    System.out.println("Thread One is alive: "
                       + o1.t.isAlive());
    System.out.println("Thread Two is alive: "
                       + o2.t.isAlive());
    System.out.println("Thread Three is alive: "
                       + o3.t.isAlive());
    // wait for threads to finish
    try {
      System.out.println("Waiting for threads to finish.");
      o1.t.join();
      o2.t.join();
```



```
      o3.t.join();
    } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }
    System.out.println("Thread One is alive: "
                       + o1.t.isAlive());
    System.out.println("Thread Two is alive: "
                       + o2.t.isAlive());
    System.out.println("Thread Three is alive: "
                       + o3.t.isAlive());

    System.out.println("Main thread exiting.");
  }
}
```

OUTPUT:

```
C:\VIJAY>javac Demojoin.java
C:\VIJAY>java Demojoin
Exception in thread "main" java.lang.NoClassDefFoundError: Demojoin
(wrong name: DemoJoin)
        at java.lang.ClassLoader.defineClass0(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:486)
        at
java.security.SecureClassLoader.defineClass(SecureClassLoader.java:111)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:248)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:56)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:195)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:188)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:297)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:286)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:253)
        at java.lang.ClassLoader.loadClassInternal(ClassLoader.java:313)
```

EXAMPLE :: 4

```
// Create a second thread by extending Thread
class NewTrd extends Thread {
```

```
NewTrd() {
 // Create a new, second thread
 super("Demo Trd");
 System.out.println("Child thread: " + this);
 start(); // Start the thread
}

// This is the entry point for the second thread.
public void run() {
 try {
  for(int i = 5; i > 0; i--) {
   System.out.println("Child Thread: " + i);
   Thread.sleep(500);
  }
 } catch (InterruptedException e) {
  System.out.println("Child interrupted.");
 }
 System.out.println("Exiting child thread.");
}
}

class ExtendThread {
 public static void main(String args[]) {
  new NewTrd(); // create a new thread
  try {
   for(int i = 5; i > 0; i--) {
    System.out.println("Main Thread: " + i);
    Thread.sleep(1000);
   }
  } catch (InterruptedException e) {
   System.out.println("Main thread interrupted.");
  }
  System.out.println("Main thread exiting.");
 }
}
```


OUTPUT:

```
C:\ >javac ExtendThread.java
C:\ >java ExtendThread
Child thread:Thread[Demo trd ,5,main]
Main thread:5
child thread:5
child thread:4
mainthread:4
child thread:3
child thread:2
mainthread:3
child thread:1
Exiting Child thread.
Main thread:2
Main thread:1
Main thread exiting
```

EXAMPLE :: 5

```
// Demonstrate thread priorities.
class Thrdprio implements Runnable {
 int click = 0;
 Thread t;
 private volatile boolean running = true;

 public Thrdprio(int p) {
  t = new Thread(this);
  t.setPriority(p);
 }

 public void run() {
  while (running) {
   click++;
  }
 }
```

```
 public void stop() {
  running = false;
 }

 public void start() {
  t.start();
 }
}

class HiLoPri {
 public static void main(String args[]) {
  Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
Thrdprio high = new Thrdprio(Thread.NORM_PRIORITY + 2);
  Thrdprio low = new Thrdprio(Thread.NORM_PRIORITY - 2);

  low.start();
  high.start();
  try {
   Thread.sleep(10000);
  } catch (InterruptedException e) {
   System.out.println("Main thread interrupted.");
  }
  low.stop();
  high.stop();

  // Wait for child threads to terminate.
  try {
   high.t.join();
   low.t.join();
  } catch (InterruptedException e) {
   System.out.println("InterruptedException caught");
  }
  System.out.println("Low-priority thread: " + low.click);
  System.out.println("High-priority thread: " + high.click);
 }
}
```


OUTPUT:

```
C:\ >javac HiLoPri.java
C:\ >java HiLoPri
Low –priority thread:0
High-priority thread:41991326
```

EXAMPLE :: 6

```
// Create multiple threads.
class NewTrd implements Runnable {
 String s; // name of thread
 Thread t;

 NewTrd(String threadname) {
  s = threadname;
  t = new Thread(this, name);
  System.out.println("New thread: " + t);
  t.start(); // Start the thread
 }
 // This is the entry point for thread.
 public void run() {
  try {
   for(int i = 5; i > 0; i--) {
    System.out.println(name + ": " + i);
    Thread.sleep(1000);
   }
  } catch (InterruptedException e) {
   System.out.println(s + "Interrupted");
  }
  System.out.println(s + " exiting.");
 }
}

class MultiThreadDemo {
 public static void main(String args[]) {
  new NewTrd("One"); // start threads
```

```
    new NewTrd("Two");
    new NewTrd("Three");

    try {
      // wait for other threads to end
      Thread.sleep(10000);
    } catch (InterruptedException e) {
      System.out.println("Main thread Interrupted");
    }

    System.out.println("Main thread exiting.");
  }
}
```

OUTPUT:

```
C:\ >javac MultiThreadDemo.java
C:\ >java MultiThreadDemo
New thread:  Thread[threadname,5,main]
New thread:  Thread [threadname,5,main]
New thread:  Thread [threadname,5,main]
Threadname: 5
Threadname: 5
Threadname: 4
Threadname: 4
Threadname: 4
Threadname: 3
Threadname: 3
Threadname: 3
Threadname: 2
Threadname: 2
Threadname: 2
Threadname: 1
Threadname: 1
Threadname: 1

Threadname exiting
Threadname exiting
```



```
Threadname exiting
Main thread exiting
```

EXAMPLE :: 7

```
// Using suspend() and resume().
class NewTrd implements Runnable {
  String s; // name of thread
  Thread t
;

  NewTrd(String threadname) {
    s = threadname;
    t = new Thread(this, name);
    System.out.println("New thread: " + t);
    t.start(); // Start the thread
  }

  // This is the entry point for thread.
  public void run() {
    try {
      for(int i = 15; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(200);
      }
    } catch (InterruptedException e) {
      System.out.println(s + " interrupted.");
    }
    System.out.println(s + " exiting.");
  }
}
class SuspendResume {
```

```
public static void main(String args[]) {
 NewTrd o1 = new NewTrd("One");
 NewTrd o2 = new NewTrd("Two");
 try {
  Thread.sleep(1000);
  o1.t.suspend();
  System.out.println("Suspending thread One");
  Thread.sleep(1000);
  o1.t.resume();
  System.out.println("Resuming thread One");
  o2.t.suspend();
  System.out.println("Suspending thread Two");
  Thread.sleep(1000);
  o2.t.resume();
  System.out.println("Resuming thread Two");
 } catch (InterruptedException e) {
  System.out.println("Main thread Interrupted");
 }
 // wait for threads to finish
 try {
  System.out.println("Waiting for threads to finish.");
  o1.t.join();
  o2.t.join();
 } catch (InterruptedException e) {
  System.out.println("Main thread Interrupted");
 }
 System.out.println("Main thread exiting.");
 }
}
```


OUTPUT:

EXAMPLE :: 8

```
// This program uses a synchronized block.
class CallmeDemo {
 void call(String msg) {
  System.out.print("[" + msg);
  try {
   Thread.sleep(1000);
  } catch (InterruptedException e) {
   System.out.println("Interrupted");
  }
  System.out.println("]");
 }
}
class Caller implements Runnable {
 String msg;
 CallmeDemo target1;
 Thread t;

 public Caller(Callme targ, String s) {
  target = targ;
  msg = s;
  t = new Thread(this);
  t.start();
 }
 // synchronize calls to call()
 public void run() {
  synchronized(target) { // synchronized block
   target.call(msg);
  }
 }
```

```
}
class Synch1 {
 public static void main(String args[]) {
  CallmeDemo target1 = new CallmeDemo();
  Caller o1 = new Caller(target1, "Hello");
  Caller o2 = new Caller(target1, "Synchronized");
  Caller o3 = new Caller(target1, "World");

  // wait for threads to end
  try {
   o1.t.join();
   o2.t.join();
   o3.t.join();
  } catch(InterruptedException e) {
   System.out.println("Interrupted");
  }
 }
}
```

OUTPUT:

```
C:\>javac Synch1.java
C:\>java Synch1
[Hello]
[Synchoronized]
[World]
```

EXAMPLE :: 9

```
// Create a second thread.
class NewTrd implements Runnable {
 Thread t;

 NewTrd() {
```



```
 // Create a new, second thread
 t = new Thread(this, "Demo Thread");
 System.out.println("Child thread: " + t);
 t.start(); // Start the thread
}

// This is the entry point for the second thread.
public void run() {
 try {
  for(int i = 5; i > 0; i--) {
   System.out.println("Child Thread: " + i);
   Thread.sleep(500);
  }  }
catch (InterruptedException e) {
  System.out.println("Child interrupted.");
 }
 System.out.println("Exiting child thread.");
}}

class ThreadDemo {
 public static void main(String args[]) {
  new NewThread(); // create a new thread
  try {
   for(int i = 5; i > 0; i--) {
    System.out.println("Main Thread: " + i);
    Thread.sleep(1000);
   }
  } catch (InterruptedException e) {
   System.out.println("Main thread interrupted.");
  }
  System.out.println("Main thread exiting.");
 }
}
```

OUTPUT:

```
C:\ >javac ThreadDemo.java
C:\ >java ThreadDemo
Child thread : Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread:5
Child Thread:4
Main Thread:4
Child Thread:3
Child Thread:2
Main Thread:3
Child Thread:1
Exiting Child Thread
Main  thread: 2
Main thread :1
Main thread exiting.
```