ARCHIVOS EN JAVA

Conceptos Básicos de Archivos

- Estructuras de Datos estudiadas:
  - Tipos: Arrays, Listas, etc.
  - Almacenadas en **memoria principal (RAM)**
    - Rápida
    - Volátil
    - Tamaño Limitado
- Para tratar grandes volúmenes de información es necesario almacenarla en **memoria secundaria** (Disco Duro, CD-ROM, Disquete, Disco USB, etc.)
- Los datos agrupados en estructuras denominadas **archivos o ficheros (File en inglés)**

Conceptos Básicos de Archivos

- Un **archivo o fichero es una colección de datos** homogéneos almacenados en un soporte físico del computador que puede ser permanente o volátil.
- Datos **homogéneos: Almacena colecciones de** datos del mismo tipo (igual que arrays/vectores)
- Cada elemento almacenado en un fichero se denomina **registro, que se compone de campos.**
- Puede ser almacenado en **diversos soportes** (Disco duro, disquete, …)

Conceptos Básicos de Archivos

- Ejemplos de archivos o ficheros:

Archivo	Registro	Campos
Biblioteca	Libro	Título, Autor, ISBN, ...
Censo	Persona	Nombre, Apellidos, DNI, Dirección, ...
Archivo de coches	Coche	Marca, Color, ...

- El tamaño de un fichero es variable puede crecer según se van insertando registros

Operaciones sobre Archivos

- Operación de **Creación**
- Operación de **Apertura. Varios modos:**
  - Sólo lectura
  - Sólo escritura
  - Lectura y Escritura
- Operaciones de **lectura / escritura**
- Operaciones de **inserción / borrado**
- Operaciones de **renombrado / eliminación**
- Operación de **desplazamiento dentro de un** fichero
- Operación de **cierre**

Operaciones sobre Archivos

- Operaciones para el manejo habitual de un fichero:

  1.- **Crearlo (sólo si no existía** previamente)

  2.- **Abrirlo**

  3.- **Operar sobre él (lectura/escritura,** inserción, borrado, etc.)

  4.- **Cerrarlo**

Tipos de Archivo:

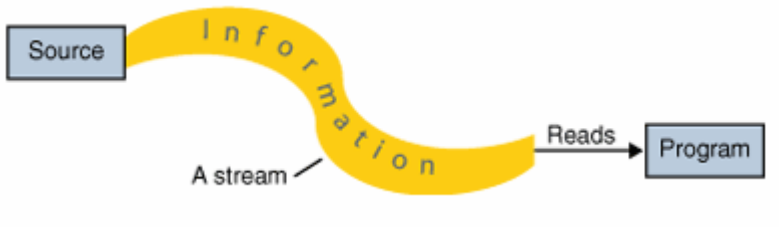
- Clasificación de los ficheros según la organización de los registros en memoria:
  - **Organización Secuencial: Registros** almacenados consecutivamente en memoria según el orden lógico en que se han ido insertando.
  - **Organización Directa o Aleatoria: El orden** físico de almacenamiento en memoria puede no coincidir con el orden en que han sido insertados.
  - **Organización Indexada. Dos ficheros:**
    - Fichero de datos: Información
    - Fichero de índice: Contiene la posición de cada uno de los registros en el fichero de datos

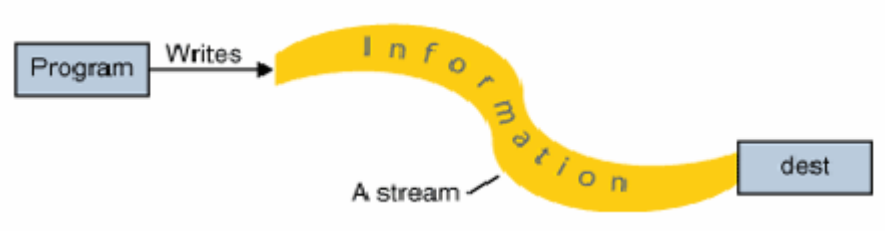
Tipos de Archivo:

- Clasificación de los ficheros según el acceso a la información almacenada:
  - **Acceso secuencial: Para acceder a un registro es** necesario pasar por todos los anteriores. Ej: Cinta de Casete
  - **Acceso directo o aleatorio: Se puede acceder a un** registro sin pasar por todos los anteriores. Ej: Disco Duro.
- Clasificación de los ficheros según el tipo de la información almacenada:
  - Ficheros **Binarios: Almacenan secuencias de dígitos** binarios (ej: ficheros que almacenan enteros, floats,…)
  - Ficheros de **Texto: Almacenan caracteres alfanuméricos en** un formato estándar (ASCII, Unicode, UTF8, UTF16, etc.). Pueden ser leídos y/o modificados por aplicaciones denominadas editores de texto (Ej: Notepad, UltraEdit, Editplus, etc.).

Conceptos básicos de Entrada/Salida

- **Streams: Canales, flujos de datos o "tuberías".** Entrada (InputStream) o Salida (OutputStream).
- Agrupados en el paquete **java.io**

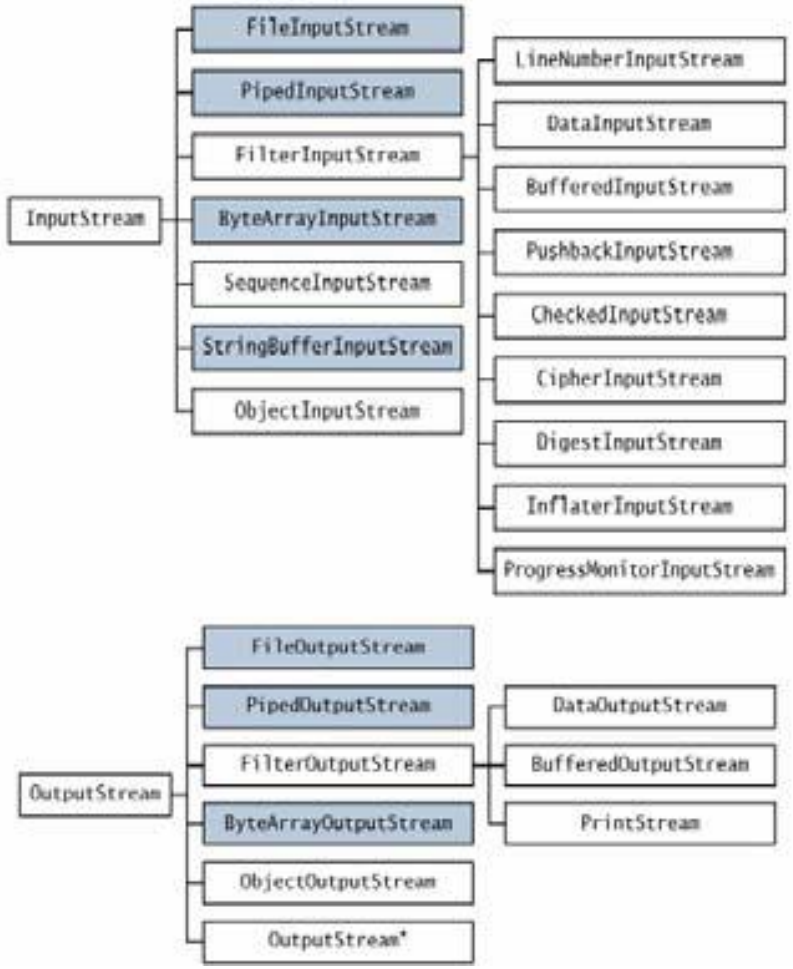




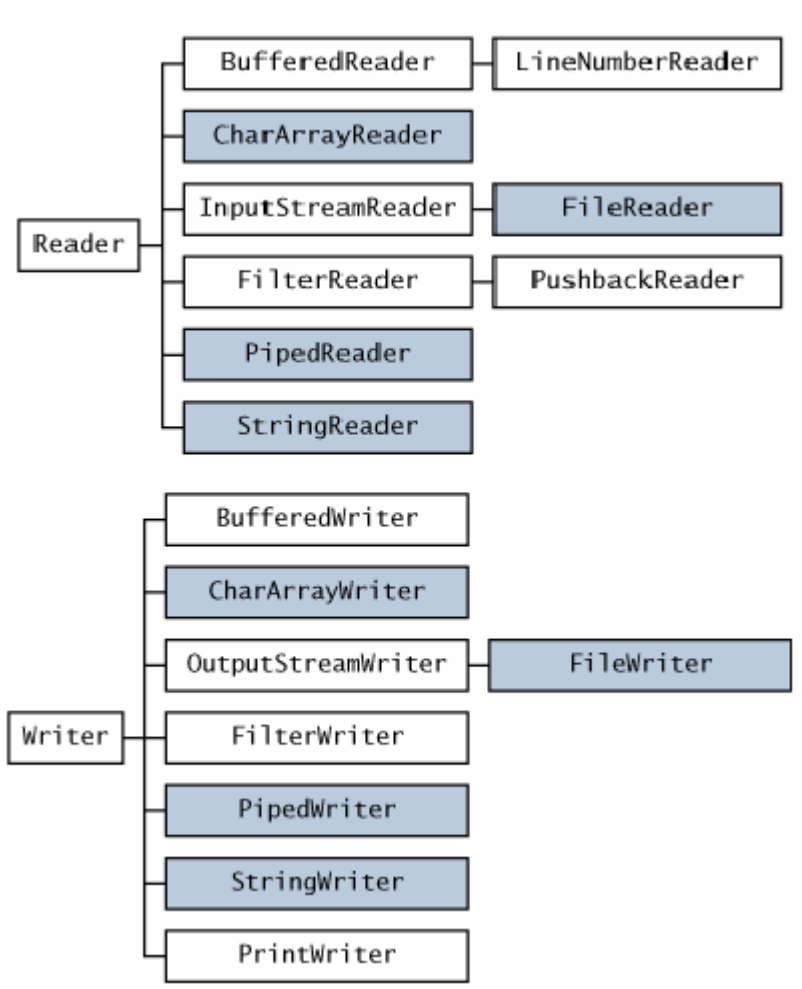
- Dos jerarquías de clases independientes, una para lectura/escritura binaria (**bytes) y otra para** lectura/escritura de caracteres de texto (**char)**

Conceptos básicos de Entrada/Salida

- Streams de bytes



- Streams de caracteres



Conceptos básicos de Entrada/Salida

- Métodos básicos de **Reader:**
  - int read()
  - int read(char cbuf[])
  - int read(char cbuf[], int offset, int length)
- Métodos básicos de **InputStream:**
  - int read()
  - int read(byte cbuf[])
  - int read(byte cbuf[], int offset, int length)

Conceptos básicos de Entrada/Salida

- Métodos básicos de **Writer:**
  - int write(int c)
  - int write(char cbuf[])
  - int write(char cbuf[], int offset, int length)
- Métodos básicos de **OutputStream:**
  - int write(int c)
  - int write(byte cbuf[])
  - int write(byte cbuf[], int offset, int length)
- Los Streams se abren automáticamente al crearlos, pero es necesario cerrarlos explícitamente llamando al método **close() cuando se dejan de usar**

Conceptos básicos de Entrada/Salida

- PrintStream / PrintWriter se utilizan para escribir cadenas de texto.
- DataInputStream / DataOutputStream se utilizan para escribir/leer tipos básicos (int, long, float,…).
- Acceso a ficheros: Según el acceso:
  - Acceso **Secuencial: El más común. Puede ser:**
    - Acceso binario: FileInputStream / FileOutputStream
    - Acceso a caracteres (texto): FileReader / FileWriter
  - Acceso **Aleatorio: Se utiliza la clase** RandomAccessFile

Conceptos básicos de Entrada/Salida

- La clase File puede usarse para representar el nombre de un **archivo concreto, o los** nombres de los archivos de un **directorio.**
- Para crear/abrir un fichero en Java se invoca a un constructor de la clase File.

```
import java.io.File;

public class CreaFile {
    public static void main(String[] args) {
        // Crea/Abre un fichero dado el nombre completo
        File f1 = new File("C:\\practicas\\ejemplo.txt");
        // Crea/Abre un fichero dado el directorio y el nombre
        File f2 = new File("C:\\practicas", "ejemplo.txt");
        // Crea/Abre un fichero dado un File con el directorio y el nombre
        File dir = new File("C:\\practicas");
        File f3 = new File(dir, "ejemplo.txt");
    }
}
```

Conceptos básicos de Entrada/Salida

```
import java.io.File;

/**
 * Dado un fichero indicado por linea de comandos, muestra si se trata de
 * un fichero, un directorio o de ninguno de los dos (no existe)
 */
public class MuestraFichero {

    public static void main(String[] args) {
        if(args.length != 1) {
            System.out.println("Debe indicar el fichero/directorio a analizar");
            System.exit(-1);
        }
        File f = null;
        try {
            f = new File(args[0]);
            if(f.isFile()) {
                System.out.println("El archivo: " + args[0] + " es un fichero");
            } else if(f.isDirectory()) {
                System.out.println("El archivo: " + args[0] + " es un directorio");
            } else {
                System.out.println("El archivo: " + args[0] + " no existe");
            }
        } catch (Exception e) {
            System.out.println("Error abriendo fichero:" + e.getMessage());
        }
    }
}
```

Archivos Binarios: FileInputStream /FileOutputStream.

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class EscribeBytes {
    public static void main(String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos.dat");
        // Crea el flujo de salida hacia el fichero
        FileOutputStream flujoSalida = new FileOutputStream(fichero);
        // Información a almacenar
        byte[] datos1 = new byte[100];
        byte[] datos2 = new byte[100];
        // Escribe datos1 byte a byte
        System.out.println("Escribiendo en fichero...");
        for(int i=0; i<datos1.length; i++) {
            datos1[i] = (byte) i;
            datos2[i] = (byte) i;
            flujoSalida.write(datos1[i]);
        }
        // Escribe datos2 de una sola vez
        flujoSalida.write(datos2);
        // Cierra el Stream, y por tanto el fichero
        flujoSalida.close();
        System.out.println("Fichero escrito");
    }
}
```

Archivos Binarios

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class LeeBytes {
    public static void main (String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos.dat");
        // Crea el flujo de entrada desde fichero
        FileInputStream flujoEntrada = new FileInputStream(fichero);
        // Calcula el tamaño del fichero
        int tamanioFichero = (int) fichero.length();
        byte[] datosLeidos = new byte[tamanioFichero];
        // Lee los datos
        System.out.println("Leyendo fichero...");
        flujoEntrada.read(datosLeidos);
        // Muestra los datos leidos por pantalla
        for(int i=0; i<datosLeidos.length; i++) {
            System.out.print(datosLeidos[i] + ", ");
        }
        System.out.println();
        // Cierra el Stream, y por tanto el fichero
        flujoEntrada.close();
        System.out.println("Fichero leido");
    }
}
```

Archivos de Datos: DataInputStream /DataOutputStream.

```
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

public class EscribeDatosPrimitivos {
    public static void main(String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos2.dat");
        // Crea el flujo de salida hacia el fichero
        FileOutputStream flujoSalida = new FileOutputStream(fichero);
        // Conecta el flujo de bytes al flujo de datos
        DataOutputStream dataOS = new DataOutputStream(flujoSalida);

        // Escribe algunos datos primitivos al fichero
        System.out.println("Escribiendo en fichero...");
        dataOS.writeBoolean(true);
        dataOS.writeChar('M');
        dataOS.writeDouble(222222222D);
        dataOS.writeInt(2006);
        dataOS.writeFloat(9999999F);
        dataOS.writeLong(42398432L);
        dataOS.writeUTF("Hola Mundo");

        // Cierra el Stream, y por tanto el fichero
        dataOS.close();
        System.out.println("Fichero escrito");
    }
}
```

Archivos de Datos

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class LeeDatosPrimitivos {
    public static void main(String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos2.dat");
        // Crea el flujo de entrada hacia el fichero
        FileInputStream flujoSalida = new FileInputStream(fichero);
        // Conecta el flujo de bytes al flujo de datos
        DataInputStream dataIS = new DataInputStream(flujoSalida);

        // Escribe algunos datos primitivos al fichero
        System.out.println("Leyendo en fichero...");
        System.out.println(dataIS.readBoolean());
        System.out.println(dataIS.readChar());
        System.out.println(dataIS.readDouble());
        System.out.println(dataIS.readInt());
        System.out.println(dataIS.readFloat());
        System.out.println(dataIS.readLong());
        System.out.println(dataIS.readUTF());

        // Cierra el Stream, y por tanto el fichero
        dataIS.close();
        System.out.println("Fichero leido");
    }
}
```

Archivos de Texto

- Escribir / Leer **cadenas de texto**
- Se utilizan las clases Reader / Writer
- PrintWriter: Hereda de Writer, permite escribir texto en un OutputStream
  - En el constructor recibe el OutputStream a utilizar
  - Tiene métodos para escribir en forma de texto todos los tipos básicos y los Strings.
  - Métodos duplicados para insertar retorno de carro al final del dato escrito (print / println)
- PrintStream es similar a PrintWriter, pero sus métodos están deprecados (la clase no, porque todavía se usa en System.out) -> Por defecto utilizar PrintWriter que es más moderna.

Archivos de Texto

```
import java.io.*;

public class EscribeTexto {
    public static void main(String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos3.dat");
        // Crea el flujo de salida hacia el fichero
        FileOutputStream flujoSalida = new FileOutputStream(fichero);
        // Conecta el flujo de bytes al flujo de datos
        PrintWriter pw = new PrintWriter(flujoSalida);

        // Escribe algunos textos al fichero
        System.out.println("Escribiendo en fichero...");
        pw.write("Hola Mundo");
        pw.write("Adios Mundo");
        pw.print("Hola otra vez");
        pw.println("Adios otra vez");
        // Cierra el Stream, y por tanto el fichero
        pw.close();
        System.out.println("Fichero escrito");
    }
}
```

Archivos de Texto

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class LeeTexto {
    public static void main(String[] args) throws IOException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos3.dat");
        // Crea el flujo de entrada hacia el fichero
        FileReader fr = new FileReader(fichero);
        // Crea un bufferedReader para leer línea a línea
        BufferedReader br = new BufferedReader(fr);

        // Lee cadenas del fichero
        System.out.println("Leyendo fichero...");
        String s = br.readLine();
        while(s != null) {
            System.out.println("Linea leida: " + s);
            s = br.readLine();
        }
        // Cierra el Stream, y por tanto el fichero
        br.close();
        System.out.println("Fichero leido");
    }
}
```

Archivos de Acceso Aleatorio

- Se utiliza la clase RandomAccessFile
- **No está basada en el concepto de flujos o Streams.**
- **No deriva de InputStream/OutputStream ni Reader/Writer**
- Permite **leer y escribir sobre el fichero, no es necesario** dos clases diferentes
- Necesario especificar el **modo de acceso al construir un** objeto de esta clase: sólo lectura o lectura/escritura
- Dispone de **métodos específicos de desplazamiento** como seek(long posicion) o skipBytes(int desplazamiento) para poder moverse de un registro a otro del fichero, o posicionarse directamente en una posición concreta del fichero

Archivos de Acceso Aleatorio

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class FicherosAccesoAleatorio {
    public static void main(String[] args) throws IOException {
        String nombre = "c:/practicas/programacion/datos4.dat";
        // Declara el fichero de acceso aleatorio
        RandomAccessFile f = new RandomAccessFile(nombre, "rw");
        // Escribe 100 bytes al fichero
        for(int i=0; i<100; i++) {
            f.write((byte) i);
        }
        // Vuelve al principio del fichero
        f.seek(0);
        // Lee uno de cada dos bytes
        for(int i=0; i<50; i++) {
            int b = f.read();
            System.out.print(b + ", ");
            f.skipBytes(1);
        }
        // Cierra el fichero
        f.close();
    }

}
```

Serialización

- **Serialización: Posibilidad de escribir/leer Objetos** java en Streams.
- Para poder serializar un objeto en java deben cumplirse los siguientes requisitos:
- Debe implementar la interfaz **Serializable (se** estudiarán las interfaces con más detenimiento en el siguiente tema).
- Todos los objetos incluidos en él tienen que implementar la interfaz **Serializable**
- Para escribir/leer objetos serializables a un Stream se utilizan las clases java ObjectInputStream /ObjectOutputStream.

Ejemplo serialización

```
import java.io.Serializable;

public class Persona implements Serializable {
    private Mano manoDerecha;
    private Mano manoIzquierda;
    private String nombre;

    public Persona(String nombre, int numDedosDerecha,
            int numDedosIzquierda) {
        this.manoDerecha = new Mano(numDedosDerecha);
        this.manoIzquierda = new Mano(numDedosIzquierda);
        this.nombre = nombre;
    }

    public String toString() {
        return "Soy una persona, me llamo " + nombre + " y tengo la" +
                " mano derecha con " + manoDerecha.numeroDedos + " dedos y la" +
                " mano izquierda con " + manoIzquierda.numeroDedos;
    }
}
```

```
import java.io.Serializable;

public class Mano implements Serializable {
    public int numeroDedos;
    public Mano(int numeroDedos) {
        this.numeroDedos = numeroDedos;
    }
}
```

Ejemplo serialización

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class EscribePersona {

    public static void main(String[] args) throws IOException {
        // Crea la persona
        Persona per = new Persona("Manuel", 5, 5);
        System.out.println(per);

        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos5.dat");
        // Crea el flujo de salida hacia el fichero
        FileOutputStream flujoSalida = new FileOutputStream(fichero);
        // Conecta el flujo de bytes al flujo de datos
        ObjectOutputStream dataOS = new ObjectOutputStream(flujoSalida);
        System.out.println("Escribiendo la persona a fichero...");
        // Escribe la persona al fichero
        dataOS.writeObject(per);
        // Cierra el fichero
        dataOS.close();
        System.out.println("Persona escrita");
    }

}
```

Ejemplo serialización

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class LeePersona {

    public static void main(String[] args) throws IOException, ClassNotFoundException {
        String ruta = "c:/practicas/programacion";
        // Declara el fichero
        File fichero = new File(ruta, "datos5.dat");
        // Crea el flujo de entrada desde el fichero
        FileInputStream flujoEntrada = new FileInputStream(fichero);
        // Conecta el flujo de bytes al flujo de datos
        ObjectInputStream dataIN = new ObjectInputStream(flujoEntrada);
        System.out.println("Leyendo la persona desde el fichero...");
        // Lee la persona del fichero
        Persona per = (Persona) dataIN.readObject();
        // Cierra el fichero
        flujoEntrada.close();
        System.out.println("Persona leida:");
        System.out.println(per);
    }
}
```