Unidad IV

Almacenamiento

de

Datos

Año 2011

> "Si nos soltaran al azar dentro del Cosmos la probabilidad de que nos encontráramos sobre un planeta o cerca de él sería inferior a una parte entre mil billones de billones (1 / 10 elevado a 33). En la vida diaria una probabilidad así se considera nula. Los mundos son algo precioso."
>
> Carl Sagan, (Cosmos)

Autor: Ing. Tymoschuk, Jorge

Unidad IV – Almacenamiento de Datos

Flujos 3
Clases File Input/Output Stream 5
Procesamiento Básico 5
Clases ByteArray Input/Output Stream 6
Clases Pipe Input/Output Stream 6
Clases Filtro 7
Clases Data Input/Output Stream 7
La Clase File, ejemplos. 10
Archivos secuenciales 12
Creación de un archivo secuencial . . .12
Consulta de un archivo secuencial . . .14
Actualización de un archivo secuencial . .16
Flujos de tipo Objeto 17
Almacenamiento de objetos de distinto tipo. .17
Tratando objetos div.: grabando, leyendo, .21
Almacenamiento en bases de datos 24
Definición, Creación y manipulación . . .24
SQL (System Query Language). 25
ACCESO A UNA BASE DE DATOS CON JDBC . . .29
Usando el editor MySQLCC 31
El proyecto DB01 36

Flujos y Archivos

En esta unidad veremos los métodos para la manipulación de archivos y directorios, así como los que se emplean para leer de, o escribir en, archivos. También estudiaremos el mecanismo de serialización de objetos mediante el cual podrá almacenar estos elementos de forma tan sencilla como si estuviera trabajando con datos de texto o numéricos.

Flujos

En esta unidad tratamos acerca de la forma de obtener información desde cualquier origen de datos capaz de enviar una secuencia de bytes y de cómo enviar información a cualquier destino que pueda también recibir una secuencia de datos. Estos orígenes y destinos de secuencias de bytes normalmente son **archivos**, aunque también podemos hablar de conexiones de red e, incluso, de bloques de memoria. Es interesante tener siempre en mente esta generalidad: por ejemplo, la información almacenada en archivos y la recuperada desde una conexión de red se manipula esencialmente de la misma forma. Desde luego, aunque los datos estén almacenados en último extremo como una secuencia de bytes, es mejor pensar en ellos como en una estructura de más alto nivel, como ser una secuencia de caracteres u objetos.

En Java, un objeto del cual podemos leer una secuencia de bytes recibe el nombre de flujo de entrada (o input stream), mientras que aquel en el que podemos escribir es un flujo de salida (u output stream). Ambos están especificados en las **clases abstractas InputStream** y **OutputStream.** Ya que los flujos orientados a byte **no** son adecuados para el procesamiento de información Unicode (recuerde que Unicode usa dos bytes para cada carácter), existe una jerarquía de clases separada para el procesamiento de estos caracteres que hereda de las **clases abstractas Reader** y **Writer.** Estas clases disponen de operaciones de lectura y de escritura basadas en los caracteres Unicode de 2 bytes, en lugar de caracteres de un solo byte.

Recuerde que el objetivo de una **clase abstracta** es ofrecer un mecanismo para agrupar el comportamiento común de clases a un nivel más alto. Esto lleva a aclarar el código y hacer el árbol de herencia más fácil de entender. La misma organización se sigue con la entrada y la salida en el lenguaje Java.

Java deriva estas cuatro clases abstractas en un gran numero de clases concretas. Veremos la mas usuales.

Un **flujo** *(stream)* es una abstracción que se refiere a un *flujo* o *corriente* de datos que fluyen entre un origen y un destino. Entre el origen y el destino debe existir una conexión o canal *(pipe)* por la que circulen los datos. La apertura de un archivo supone establecer la conexión del programa con el dispositivo que contiene al archivo; por el canal que comunica el archivo con el programa fluyen las secuencias de datos. Abrir un archivo supone crear un objeto quedando asociado con un flujo.

Al comenzar la ejecución de un programa Java se crea automáticamente un objeto de la clase java.lang.**System.** Esta clase incluye estaticamente tres atributos que son **objetos flujo** de las clases abajo indicadas.

static PrintStream	**err** The "standard" error output stream.
static InputStream	**in** The "standard" input stream.
static PrintStream	**out** The "standard" output stream.

System.err: Objeto para salida estándar de errores por pantalla.
System.in: Objeto para entrada estándar de datos desde el teclado.
System.out: Objeto para salida estándar de datos por pantalla.

Estos tres objetos sirven para procesar secuencias de caracteres en modo texto. Así, cuando se ejecuta System. out. print ("Aquí me pongo a cantar"); se escribe la secuencia de caracteres en pantalla, y cuando se ejecuta System. in. read () se capta un carácter desde teclado.

Todo esto lo hemos visto y usado extensamente.

Si el archivo se abre para salida, usaremos un flujo de salida. Si el archivo se abre para entrada, necesitamos un flujo de entrada. Los programas leen o escriben en el flujo, que puede estar conectado a un dispositivo o a otro. El flujo es, por tanto, una abstracción, de tal forma que las operaciones que realizan los programas son sobre el flujo independientemente del dispositivo al que esté asociado.

El paquete **java.io** agrupa el conjunto de clases para el manejo de entrada y salida con archivos:
Siempre que se vaya a procesar un archivo se tienen que utilizar clases de este paquete, por lo que se debe de importar: import java. io.* si queremos el conjunto de todas las clases.

Todo el proceso de entrada y salida en lava se hace a través de flujos (stream). Los flujos de datos, de caracteres, de bytes se pueden clasificar en flujos de entrada (**Input Stream**) y en flujos de salida (**OutputStream**). Por ello Java declara dos clases abstractas que declaran métodos que deben redefinirse en sus clases derivadas. InputStream es la clase base de todas las clases definidas para streams de entrada, y OutputStream es la clase base de todas las clases de stream de salida.

Extienden **abstract InputStream**: FilelnputStream, ByteArraylnputStream, PipelnputStream, SequencelnputStream, StringBufferlnputStream, FilterlnputStream

Extienden **abstract OutputStream**: FileOutputStream, ByteArrayOutputStream, PipeOutputStream, FilterOutputStream

Clase FileInputStream

La clase FilelnputStream se utiliza para leer bytes desde un archivo. Proporciona operaciones básicas para leer un byte o una secuencia de bytes.

FilelnputStream(String nombre) throws FileNotFoundException¡
Crea un objeto inicializado con el nombre de archivo que se pasa como argumento.

FilelnputStream(File nombre) throws FileNotFoundException¡
Crea un objeto inicializado con el objeto archivo pasado como argumento.

int read() throws IOException¡
Lee un byte del flujo asociado. Devuelve -1 si alcanza el fin del archivo.

int read(byte[] s) throws IOException¡
Lee una secuencia de bytes del flujo y se almacena en el array s. *Devuelve* -1 *si alcanza el fin del*
archivo, o bien el número de bytes leídos.

int read(byte[] s, int org, int len) throws IOException¡
Lee una secuencia de bytes del flujo y se almacena en el array s desde la posición org y un máximo de len bytes. Devuelve -1 si alcanza elfin del archivo. o bien el número de bytes leídos.

Clase FileOutputStream

Con la clase FileOutputStream se pueden escribir bytes en un flujo de salida asociado a un archivo.

FileOutputStream(String nombre) throws IOException¡
Crea un objeto inicial izado con el nombre de archivo que se pasa como argumento.

FileOutputStream(String nombre, boolean sw) throws IOException¡
Crea un objeto inicializado con el nombre de archivo que se pasa como argumento. En el caso de que sw = true los bytes escritos se añaden al final.

FileOutputStream(File nombre) throws IOException¡
Crea un objeto inicializado con el objeto archivo pasado como argumento.

void write(byte a) throws IOException¡ *Escribe el byte a en el flujo asociado.*

void write(byte[] s) throws IOException¡ *Escribe el array de bytes en el flujo.*

void write(byte[] s, int org, int len) throws IOException¡
Escribe el array s desde la posición org y un máximo de len bytes en el flujo.

Procesamiento Basico

Ejemplos usando FileInputStream y FileOutputStream

// La aplicación crea un objeto stream de salida, el archivo se denomina cadena.txt.
// El programa graba varios arrays de bytes inicializados desde cadenas,
// Para tratar las excepciones se define un bloque try y un catch de captura.

```
Grabadas 4 lineas (exito)
Process Exit...
```

```
import java.io.*;
class ArchivoCad{
	private int contLin = 0;
	public void demo(){
		String str = new String();
		String lineas[] = 	{"La mas antigua de todas las filosofias",
					 "la de la evolucion",
					 "estuvo maniatada de manos y pies",
					 "y relegada a la oscuridad mas absoluta ..."};
		byte [] s;
		try {
			FileOutputStream f = new FileOutputStream("cadena.txt");
			for(int i = 0; i < lineas.length;i++){
				s = lineas[i].getBytes(); // copia la cadena en el array de bytes s;
				f.write(s);			// graba el array de bytes
				f.write((byte)'\n') ;	// graba el avance de carro
				contLin++;			// cuenta
			}
		}
		catch (IOException e){System.out.println("Problema grabacion");}
		finally {System.out.println("Grabadas "+contLin+" lineas (exito)");}
	}
```

```
        public static void main(String [] args){
                ArchivoCad arch = new ArchivoCad();
                arch.demo();
        }
}
```

Nota: Si el archivo ya existía, es recubierto por el nuevo, sin aviso ni excepcion de ninguna clase

```
//  Al archivo ya creado queremos agregarle unas líneas, al final.
//  Todo lo que necesitamos es un segundo parámetro en  new FileOutputStream("cadena.txt", true);
import java.io.*;
class ArchivoCad02{
        private int contLin = 0;
        public void demo(){
                String str = new String();
                String lineas[] =  {"durante el milenio del escolasticismo teologico.",
                                "Pero Darwin infundio nueva savia vital en la antigua estructura;",
                                "las ataduras saltaron, y el pensamiento revivificado",
                                "de la antigua Grecia ha demostrado ser una expresion mas",
                                "adecuada del orden universal ... T.H.HUXLEY, 1887"};
                byte [] s;
                try {
                        FileOutputStream f = new FileOutputStream("cadena.txt",true);
                        for(int i = 0; i < lineas.length;i++){
                                s = lineas[i].getBytes(); // copia la cadena en el array de bytes s;
                                f.write(s);                              // graba el array de bytes
                                f.write((byte)'\n') ;      // graba el avance de carro
                                contLin++;                               // cuenta
                        }
                }
                catch (IOException e){System.out.println("Problema grabacion");}
                finally {System.out.println("Agregadas "+contLin+" lineas (exito)");}
        }
        public static void main(String [] args){
                ArchivoCad02 arch = new ArchivoCad02();
                arch.demo();
        }
}
```

```
Agregadas 5 lineas (exito)
Process Exit...
```

```
La mas antigua de todas las filosofias
la de la evolucion
estuvo maniatada de manos y pies
y relegada a la oscuridad mas absoluta ...
durante el milenio del escolasticismo teologico.
Pero Darwin infundio nueva savia vital en la antigua estructura;
las ataduras saltaron, y el pensamiento revivificado
de la antigua Grecia ha demostrado ser una expresion mas
adecuada del orden universal ... T.H.HUXLEY, 1887
Lectura terminada !!!
Process Exit...
```

```
//  queremos leer

import java.io.*;
class ArchivoLee {
   public void demo()
      int c;
      try {
         FileInputStream f = new FileInputStream("cadena.txt");
         while ((c = f.read()) !=-1)
             System.out.print((char)c) ;
      }
      catch(IOException e) {
          System.out.println("Anomalía flujo de entrada "+
            "(Revisar si el archivo existe) .");}
      finally {System.out.println("Lectura terminada !!!");}
   }
   public static void main(String [] args) {
      ArchivoLee lectura = new ArchivoLee();
```

```
      lectura.demo();
   }
}
```

Hemos visto ejemplos concretos de tratamiento de archivos de texto usando las clases **FileOutputStream** y **FileInputStream.** En Java las posibilidades de tratamiento de archivos son muy variadas. En nuestra asignatura pretendemos ver con algun detalle el tratamiento de archivos secuenciales y de acceso directo, por lo que solo citaremos a título informativo estas otras alternativas.

Clases ByteArrayInputS tream y ByteArrayOutputStream
Estas clases permiten asociar un flujo con un array de bytes, en vez de un archivo. Usandolas, podemos hacer que un objeto stream de entrada lea del array de bytes, y que un objeto stream de salida escriba en un array de bytes interno que crece dinámicamente.

Clases PipeInputStream y PipeOutputStream
Estas clases se utilizan para transferir datos entre tareas *(threads)* sincronizadas. Permite a dos tareas comunicarse mediante llamadas a los métodos de escritura y lectura. Se ha de definir un objeto stream de tipo *Pipelnput* y otro objeto flujo de tipo *PipeOutput.* Para enviar datos a una tarea, el objeto flujo de salida invoca a la operación wri te ( ). La tarea que recibe datos los captura a través del objeto flujo de entrada, llamando a métodos de lectura, read() y receive( ). Ambas clases tienen un constructor al que se le pasa como argumento el objeto *pipe* de la otra clase; también tienen el método connect(), que tiene como argumento el objeto *pipe* con el que se conectan.

Clases Filtro
Los elementos transferidos, de entrada o de salida, con los flujos utilizados anteriormente han sido siempre secuencias de bytes. Los flujos *filtro* también leen secuencias de bytes, pero organizan internamente estas secuencias para formar datos de los tipos primitivos (int, long, double. . .) . Los stream *filtro* son una abstracción de las secuencias de bytes para hacer procesos de datos a más alto nivel; con esta abstracción ya no tratamos los items como secuencias o *«chorros»* de bytes, sino de forma elaborada con más funcionalidad. Así, a nivel lógico, se pueden tratar los datos dentro de un buffer, escribir o leer datos de tipo int, long, double directamente y no mediante secuencias de bytes. Los objetos stream *filtro* leen de un flujo que previamente ha tenido que ser escrito por otro objeto stream *filtro* de salida.
Considerando de forma aislada la jerarquía de clase stream *filtro* hay dos clases base: **FilterlnputStream** y **FilterOutputStream** (derivan directamente de InputSream y OutputStream). Son clases abstractas y por consiguiente la interfaz de cada clase tiene que ser definida en las clases derivadas.

La clase **FilterlnputStream** es extendida por BufferedlnputStream, LineNumberlnputStream, PushbacklnputStream y DataInputStream.

La clase **FilterOutputStream** es extendida por BufferedOutputStream, DataOutputStream y PrintStream.

Clases DataInputStream y DataOutputStream
La clase para entrada, DatalnputStream, *filtra* una secuencia de bytes, los organiza, para poder realizar lecturas de tipos de datos primitivos directamente: char, byte, short, int, long, float, double, boolean. Un objeto de esta clase lee de un flujo de entrada de bajo nivel (flujo de bytes) al que está asociado. La asociación se realiza en dos pasos:

1) - FileInputStream gs = new FileInputStream(“Palabras.dat");

```
      //  Generamos objeto FileInputStream
2) - DataInputStream ent = new DataInputStream(gs);
      //  Generamos objeto DataInputStream
```

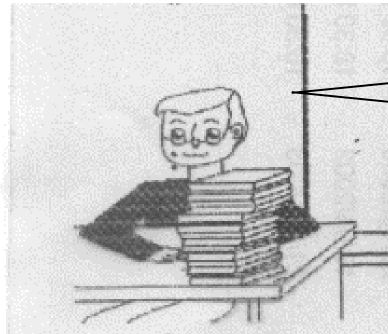
Algunos de los métodos más importantes de **DataInputStream**, todos son public y final.

```
DataInputStream(InputStream entrada)
// Constructor, requiere de parámetro tipo FileInputStream .

boolean readBoolean{) throws IOException
// Devuelve el valor de tipo boolean leído.

byte readByte() throws IOException // Devuelve el valor de tipo byte leído.

short readShort() throws IOException // Devuelve el valor de tipo short leído.

char readChar{) throws IOException       // Devuelve el valor de tipo char leído.

int readlnt() throws IOException         // Devuelve el valor de tipo int leído.

long readLong() throws IOException       // Devuelve el valor de tipo long leído.

float readFloat() throws IOException     // Devuelve el valor de tipo float leído.

double readDouble() throws IOException   // Devuelve el valor de tipo double
leído.

String readUTF() throws IOException
// Devuelve una cadena que se escribió en formato UTF.

String readLine() throws IOException // Devuelve la cadena leída hasta fin de
línea.
```

Algunos de los métodos más importantes de **DataOutputStream**, todos son public y final.

```
DataOutputStream(OutputStream destino) // Constructor, requiere de parámetro tipo FileOutputStream
void writeBoolean(boolean v) throws IOException // Escribe v de tipo boolean

void writeByte(int v) throws IOException // Escribe v como un byte.

void writeShort(int v) throws IOException // Escribe v como un short.
void writeChar(int v) throws IOException // Escribe v como un carácter.
void writeInt(int v) throws IOException  // Escribe  int v.
void writeLong(long v) throws IOException // Escribe el dato de tipo long v.
void writeFloat(float v) throws IOException // Escribe el dato de tipo float v.
void writeDouble(double v) throws IOException  // Escribe el valor de tipo double
v.
void writeUTF(String cad) throws IOException   // Escribe cadena cad en formato
UTF.
int size()                                     // Devuelve el tamaño del flujo.
```

Profe, esto está un poco denso … No tiene un ejemplo, algo sencillito, algo …



Sencillito… si, y claro. Recuerdan la clase ArrItems? Su constructor nos generaba un arrays de ítems código/valor, podriamos generar unos viente y grabarlos. Luego leerlos… Les parece?

Grabando veinte elementos de Arritems

```
import ArrItems;
class GrabItems{
        private ArrItems aItems;
        private int itGrb;
        public GrabItems(int tam, char tipo){
                aItems = new ArrItems(tam,tipo);
                itGrb = 0;
        }

        public void demo()throws IOException{
                FileOutputStream f = new FileOutputStream("Archivo Items.tmp");
                DataOutputStream strm = new DataOutputStream(f) ;
                for(int i = 0; i <  aItems.talle;i++){
                        strm.writeInt(aItems.item[i].getCodigo());
                        strm.writeFloat(aItems.item[i].getValor());
                        itGrb++;
                }
                System.out.println("Grabados "+itGrb+" items");
        }

        public static void main(String[] args) throws IOException {
                GrabItems grbIt = new GrabItems(20,'R');
                grbIt.demo();
        }
}
```

```
Primeros 10 de 20
 elementos Item
1 - 16.220732
2 - 3.0816941
19 - 18.192905
6 - 9.128724
17 - 12.868467
11 - 3.3735917
1 - 7.5119925
4 - 7.7511096
4 - 14.572884
3 - 10.45162
Grabados 20 items
Process Exit...
```

Leyendo "Archivo Ítems.tmp"

```
import java.io.*;
class LeerItems{
        private int itLei;
        public LeerItems(){
                itLei = 0;
        }
        public String toString(){
                FileInputStream f;
                DataInputStream strm;
                int cod;
                float val;
                String aux = "Valores leidos \nArchivo Items.tmp\n";
                try{
                        f = new FileInputStream("Archivo Items.tmp");
                        strm = new DataInputStream(f);
                        while (true){
                                cod = strm.readInt();
                                val = strm.readFloat();
```

```
Valores leidos
Archivo Items.tmp
1 - 16.220732
2 - 3.0816941
19 - 18.192905
6 - 9.128724
17 - 12.868467
. . . . .
Otros 12 items
. . . . .
9 - 6.0928297
8 - 16.284492
1 - 1.2296109
lectura terminada
Leidos 20 items
Process Exit...
```

```
                    aux+= cod+" - "+val+"\n";
                    itLei++;
                }
                    }
            catch(EOFException eof){aux+="lectura terminada \n";}
            catch(IOException io){aux+="Anomalia al procesar flujo";}
            finally{aux+="Leidos "+itLei+" items \n";}
            return aux;
        }
        public static void main(String[] args) throws IOException {
            LeerItems leeIt = new LeerItems();
            System.out.println(leeIt);
        }
}
```

Nota: Observese que fin de archivo de entrada es tratado como excepción. De ello se ocupa **EOFException** (End Of File Exception)

La clase File

En todos los ejemplos de los apartados anteriores, para crear un archivo se ha instanciado un flujo de salida y se ha pasado una cadena con el nombre del archivo. Sin embargo, todos los constructores de clases de flujo de salida que esperan un archivo pueden recibir un objeto FILE con el nombre del archivo y más propiedades relativas al archivo. La clase FILE define métodos para conocer propiedades del archivo (última modificación, permisos de acceso, tamaño...); también métodos para modificar alguna característica del archivo.
Los constructores de FILE permiten inicializar el objeto con el nombre de un archivo y la ruta donde se encuentra. También, inicializar el objeto con otro objeto FILE como ruta y el nombre del archivo.

public File(String nombreCompleto)
  Crea un objeto File con el nombre y ruta del archivo pasado como argumento.

public File(String ruta, String nombre)
  Crea un objeto File con la ruta y el nombre del archivo pasado como argumento.

public File(File ruta, String nombre)
Crea un objeto File con un primer argumento que a su vez es un objeto File con la ruta y el nombre del archivo como segundo argumento.

Algunos ejemplos:

File mi Archivo = new File("C:\LIBRO\Almacen.dat");
Crea un objeto File con el archivo Almacen.dat que está en la ruta C\LIBRO;

File otro = new File("COCINA" , "Enseres.dat");
Crea el objeto otro *con el archivo Enseres.dat que está en la misma carpeta que COCINA;*

File dir = new File ("C: \JAVA \EJERCICIOS") ;
File fil = new File(dir,"Complejos.dat");
Crea el objeto dir con un directorio o ruta absoluta. A continuación crea el objeto fil con el archivo Complejos. dat que está en la ruta especificada por el objeto dir.

Observación: La ventaja de crear objetos File con el archivo que se va a procesar es que podemos hacer controles previos sobre el archivo u obtener información sobr él..

Información sobre un archivo:

public boolean exists() *Devuelve true si existe el archivo (o el directorio).*

public boolean canWrite() *Devuelve true si se puede escribir en él, caso contrario es de sólo lectura.*

public boolean canRead() *Devuelve true si es de sólo lectura.*

public boolean isFile() *Devuelve true si es un archivo.*

public boolean isDirectory() *Devuelve true si el objeto representa a un directorio.*

public boolean isAbsolute() *Devuelve true si el directorio es la ruta completa.*

public long length() *Devuelve su tamaño en bytes. Si es directorio devuelve cero.*

public long lastModified() *Devuelve la hora de la última modificación.*

Un ejemplo usando class File: Chequearemos algunos de los métodos que acabamos de enumerar.

```
import java.io.*;
class Atributos {
  public void demo(){
    File miObj;         // Mi objeto
    BufferedReader entrada = new BufferedReader(
      new InputStreamReader(System.in));

    //Construyo entrada, objeto tipo BufferedReader,
    //su constructor necesita de otro objeto, del tipo
    //InputStrearnReader, cuyo parametro debe ser de
    //tipo System.in (texto)

    String cd;
    System.out.println("Informe nombre de archivo");
    System.out.println("o bien un directorio...");
    System.out.println("Termina con línea vacía. ");
    try{
      do{
         cd = entrada.readLine();
         miObj = new File(cd);
         if (miObj.exists()){
            System.out.println(cd + " existe !!!");
            if (miObj.isFile()){
               System.out.println(" Es un archivo,");
               System.out.println(" su tamaño: " + miObj.length());
            }
            else if (miObj.isDirectory())
                    System.out.println(" Es un directorio. ");
                  else
                    System.out.println(" Existe, desconocido...");
         }  //    verdadero de if (miObj.exists())
         else
            if (cd.length() > 0)
                System.out.println(" No existe !");
      } while (cd.length() > 0);
    }
    catch(IOException e){System.out.println("Excepción " + e.getMessage());}
  } //demo

  public static void main(String[] args){
      Atributos atr = new Atributos();
```

```
Informe nombre de archivo
o bien un directorio...
Termina con línea vacía.
d:\tymos
 No existe !
atributos.class
 atributos.class existe !!!
 Es un archivo,
 su tamaño: 1629
atributos.java
 atributos.java existe !!!
 Es un archivo,
 su tamaño: 1357
e:\tymos\catedras
 e:\tymos\catedras existe !!!
 Es un directorio.

Process Exit...
```

```
      atr.demo();
  }
}
```

mas métodos de File

public String getName() // retorna *nombre del archivo o del directorio del objeto*
public String getpath() // *Devuelve la ruta relativa al directorio actual.*
public String getAbsolutePath() // *Devuelve la ruta completa del archivo o directorio.*
public boolean setReadOnly() // *Marca el archivo como de sólo lectura.*
public boolean delete() // *Elimina el archivo o directorio (debe estar vacío)*
public boolean renameTo(File nuevo) Renombra objeto File a nuevo.
public boolean mkdir() // *Crea el directorio con el que se ha creado el objeto.*
public String[] list() // *Devuelve un array de cadenas, contiendo elementos (archivo o directorio) pertenecientes al directorio en el que se ha inicializado el objeto.*

Archivos Secuenciales

La organización de un archivo define la forma en la que los registros se disponen sobre el soporte de almacenamiento, o también se define la organización como la forma en que se estructuran los datos en un archivo. En general se consideran tres organizaciones fundamentales:

· Organización secuencial.
· Organización directa o aleatoria.
· Organización secuencial indexada.

Un archivo con organización secuencial es una sucesión de registros almacenados consecutivamente sobre el soporte externo, de tal modo que para acceder a un registro *n* dado es obligatorio pasar por los n-l registros que le preceden.

En un archivo secuencial los registros se insertan en el archivo en orden de llegada, es decir, un
registro de datos se almacena inmediatamente a continuación del registro anterior.

Las operaciones básicas que se permiten en un archivo secuencial son: *escribir su contenido, aña*
dir un registro (al final del archivo) y *consultar registros.*

Java procesa los archivos como secuencias de bytes, no determina la estructura del archivo. El programador es el que determina el tipo de tratamiento del archivo, si va a ser secuencial o de acceso directo. Primero debe especificar el concepto de registro según los datos que se van a almacenar; Java tampoco contempla la estructura de registro. Los campos en que se descompone cada registro, según el programador determine, se han de escribir uno a uno. Además, la operación de lectura del archivo creado tiene que hacerse de forma recíproca, si por ejemplo el primer campo escrito es de tipo entero, al leer el método de lectura debe ser para un entero. Cuando se terminan de escribir todos los reistros se cierra el flujo (close())

Creación del archivo secuencial Corredores.dat

El proceso de creación de un archivo secuencial es también secuencial, los registros se almacenan consecutivamente en el mismo orden en que se introducen.
El siguiente programa va a crear un archivo secuencial en el que se almacenan registros relativos a los corredores que participan en una carrera.

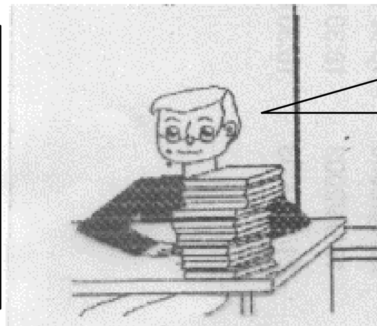
```
import java.io.*;
import In;
```

```
class Corredor{
      // Datos del corredor
      protected String nombre;
      protected int edad;
      protected char categoria;
      protected char sexo;
      protected int minutos;
      protected int segundos;

      // Datos del archivo
      protected File file;
      protected FileOutputStream miArch;
      protected DataOutputStream salida;
      protected BufferedReader entrada;

      public Corredor(){           // Constructor
            miArch  = null;   // Inicializando referencias
            entrada = null;
            salida    = null;
            file        = nul;
      }

      public boolean leerDatos()throws IOExcepti
            System.out.print("Nombre: ");
            nombre = In.readLine();
            if (nombre.length() > 0){
                  System.out.print("Edad: ");
                  edad = In.readInt();
                  if (edad >= 40) categoria = 'V
                  else              categoria = '
                  System.out.print("Sexo,(M,F):
                  sexo = In.readChar();
                  System.out.print("Minutos:  ")
                  minutos = In.readInt();
                  System.out.print("Segundos: ")
                  segundos = In.readInt();
                  return true;
            }
            else return false;
      }
      // método para escribir en el flujo
      public void grabarDatos () throws IOExcept
            salida.writeUTF(nombre);
            salida.writeInt(edad);
            salida.writeChar(categoria);
            salida.writeChar(sexo);
            salida.writeInt(minutos);
            salida.writeInt(segundos);
      }

      public void crearArchivo(){
            System.out.println("Generando archivo de corredores");
            System.out.print("Nombre archivo: ");
            String nomeArch = In.readLine();
            nomeArch+=".dat";
            file    = new File(nomeArch);      //    Instanciando objeto File
            if (file.exists())
                  System.out.println("(Ok: Archivo creado)");
            try{
                  miArch = new FileOutputStream(file);
                  salida = new DataOutputStream(miArch);
                  entrada = new BufferedReader(
                              new InputStreamReader(System.in));
            }
```

```
Generando archivo de corredores

Nombre archivo: Corredores
(Ok: Archivo creado)

Informe datos del corredor
Para terminar, Nombre:<ENTER>

Nombre: Primero
Edad: 22
Sexo,(M,F): M
Minutos:  20
Segundos: 33
Nombre: Segundo
Edad: 21
Sexo,(M,F): M
Minutos:  22
Segundos: 50
Nombre: Tercero
Edad: 25
Sexo,(M,F): M
Minutos:  22
Segundos: 55
Nombre: Cuarto
Edad: 33
Sexo,(M,F): F
Minutos:  40
Segundos: 10
Nombre:
Process Exit...
```

```
            catch (IOException e){System.out.println("Excepción creando archivo");}
            System.out.println("Informe datos del corredor");
            System.out.println("Para terminar, Nombre:<ENTER>");
            ile = new File("Corredores.dat");
            try{
                  boolean mas = true;
                  miArch = new FileOutputStream(file);
                  salida = new DataOutputStream(miArch);
                  BufferedReader entrada = new BufferedReader(
                              new InputStreamReader(System.in));
                  while (mas){
                        mas = leerDatos();
                        if (mas) grabarDatos();
                  }
                  miArch.close();
            }
            catch       (IOException       e){System.out.println("Excepción       de
Entrada/Salida");}
      }

      public static void main(String[] args){
            Corredor corr = new Corredor();
            corr.crearArchivo();
      }
}
```

Profe, Ud. disculpe, .pero esa clase Corredores está muy incompleta… Mínimamente debería tener comportamiento para leer el archivo y satisfacer alguna consulta. Sugiero, por ejemplo, poder informar cuales y cuantos corredores empatan en tantos minutos …



No me diga. Digamos que es una clase que es apenas un comienzo, está todo por hacer … Haremos lo que Ud pide, ya. Hum… Solo que en vez de modificar Corredores, la extendemos en una nueva clase que incorpore lo que Ud pide. Y, según opción del usuario, podremos crear el archivo o consultarlo … Le parece bien?

Consulta de un archivo secuencial

El proceso de consulta de una información en un archivo de organización secuencial se debe efectuar obligatoriamente en modo secuencia!. Por ejemplo, si se desea consultar la información contenida en el registro 50, se deberán leer previamente los 49 registros que le preceden. En el caso del archivo Corredores, si se desea buscar el tiempo de carrera de un participante del que se conoce su nombre y edad, será necesario recorrer todo el archivo desde el principio hasta encontrar el registro buscado o leer el último registro (fin de archivo).

En Java, para leer un archivo secuencial es necesario conocer el diseño que se ha hecho de cada registro, cómo se han escrito los campos en el proceso de creación. Los métodos que se utilicen para leer tienen que corresponderse con los métodos que se utilizaron para escribir. Así, si se escribió una cadena con el método writeUTF (), se ha de leer con readUTF (); si se escribió un entero con writeInt () , se leerá con readInt () , y así con cada tipo de dato.

```
import java.io.*;
import Corredor;
import In;
class CorreCaminos extends Corredor{
      protected char opcion;
```

```
protected int minutes;
// Datos del archivo
FileInputStream  miArch;
DataInputStream entrada;

public CorreCaminos(){
      super();
}

public      boolean existeArchivo(){
      System.out.println("Consultando archivo de corredores");
      System.out.print("Nombre archivo: ");
      String nomeArch = In.readLine();
      nomeArch+=".dat";
      file    = new File(nomeArch);      //    Instanciando objeto File
      if (!file.exists()){
            System.out.println("No encuentro "+nomeArch);
            return false;
      }
      try{
            miArch = new FileInputStream(file);
            entrada = new DataInputStream(miArch);
      }
      catch (IOException e){
            System.out.println("Excepción crea
            return false;
      }
      return true;
}

public void consulta(){
      System.out.println("Consulta por empatad
      while (true){
            System.out.print("Cuantos minutos
            minutes = In.readInt();
            if (minutes > 0)break;
      }
      System.out.println(this);
}

public String toString(){
      boolean finArch = false;
      int cont = 0, cEmp = 0;
      String aux = "Corredores empatados\n";
      while (!finArch){
            try{
                  nombre            = entrada.readUTF();
                  edad              = entrada.readInt();
                  categoria   = entrada.readChar();
                  sexo        = entrada.readChar();
                  minutos           = entrada.readInt();
                  segundos    = entrada.readInt();
            }
            catch(EOFException eof){finArch = true;}
            catch(IOException io){aux+="Anomalia al procesar flujo";}
            if (!finArch){
                  cont++;
                  if (minutos == minutes){
                        aux+=nombre+" , "+minutos+" , "+segundos+"\n";
                        cEmp++;
                  }
            }
      }
      aux+= "lectura terminada \n";
```

```
Por favor, opte:

Crear  archivo: 1

Consultarlo . . 2

Salir . . <Enter>2
Consultando     archivo     de
corredores

Nombre archivo: Corredores
Consulta por empatados

Cuantos minutos ? 22
Corredores empatados
Segundo , 22 , 50
Tercero , 22 , 55
lectura terminada
Leidos 4, empat. 2

Process Exit...
```

```
Por favor, opte:

Crear  archivo: 1

Consultarlo . . 2
Salir . . <Enter>2
Consultando     archivo     de
```

```
            aux+="Leidos "+cont+", empat. "+cEmp+"\n";
            return aux;
      }

      public void queHago(){
            System.out.println("Por favor, opte:");
            System.out.println("Crear  archivo: 1");
            System.out.println("Consultarlo . . 2");
            System.out.print("Salir . . <Enter>");
            opcion = In.readChar();
            if (opcion == '1'){     // Debemos crear el archivo
                  crearArchivo();
            }
            if (opcion == '2'){
                  if (existeArchivo())
                        consulta();
            }
      }           //    public void queHago

      public static void main(String[] args){
            CorreCaminos corre = new CorreCaminos();
            corre.queHago();
      }
}
```

Actualización de un archivo secuencial

La actualización de un archivo con organización secuencial supone añadir nuevos registros, modificar datos de registros existentes o borrar registros; es lo que se conoce como *altas, modificaciones* y *bajas.*

El proceso de dar de alta un determinado registro o tantos como se requiera se puede hacer al inicializar el flujo de la clase FileOutputStream, pasando true como segundo argumento al constructor (el primer argumento es el archivo). De esta forma, el nuevo registro que se da de alta se añade al final del archivo.
Así, por ejemplo, para añadir nuevos registros al archivo Corredores:

```
File f = new File("Corredores.dat");
FileOutputStream mf = new FileOutputStream(f,true);
DataOutputStream fatI = new DataOutputStream(mf) ;
```

Para dar de baja a un registro del archivo secuencial se utiliza un archivo auxiliar, también secuencial. Se lee el archivo original registro a registro y en función de que coincida o no con el que se quiere dar de baja se decide si el registro debe ser escrito en el archivo auxiliar. Si el registro se va a dar de baja, se omite la escritura en el archivo auxiliar. Si el registro no se da de baja, se escribe en el archivo auxiliar. Una vez que termina el proceso completo con todo el archivo, se tienen dos archivos: el original y el auxiliar. El proceso de baja termina cambiando el nombre del archivo auxiliar por el del original. Los flujos que se deben crear son:

```
File fr = new File("Original");
FileInputStream fo = new FileInputStream(fr) ;
DataInputStream forg = new DataInputStream(fo) ;

// para el archivo auxiliar
File fx = new File("Auxiliar");
FileOutputStream fa = new FileOutputStream(fx) ;
DataOutputStream faux = new DataOutputStream(fa) ;
```

Con la llamada a los métodos de la clase File: delete () y renameTo () , para

eliminar el archivo original una vez procesado y cambiar de nombre el archivo auxiliar, se termina el proceso de dar de baja.

Las actualizaciones de registros de un archivo secuencial se pueden hacer siguiendo los mismos pasos que dar de baja. Los registros del archivo original se leen y se escriben en el archivo auxiliar, hasta alcanzar el registro a modificar. Una vez que se alcanza éste, se cambian los datos o campos deseados y se escribe en el archivo auxiliar. El archivo original se procesa hasta leer el último registro; se termina borrando el archivo original y cambiando el nombre del auxiliar por el nombre del original.
NOTA: Las actualizaciones de un archivo secuencial con movimientos de altas, bajas y modificaciones (ABM) requieren que ambos archivos estén ordenados idénticamente, por los mismos campos. La lógica del programa que trata estas actualizaciones no es trivial. Normalmente esta problemática se resuelve con una base de datos y estas actualizaciones se declaran en en lenguaje SQL (Update myBase …)

Flujos de tipo objeto

El uso de registros de longitud fija es una buena elección cuando necesite almacenar datos del mismo tipo. Sin embargo, los objetos creados en un programa OOP raramente lo serán. Por ejemplo, puede tener un array llamado staff que, nominalmente, es un array de registros Empleados pero que contiene objetos que son instancias de una clase hija como Gerentes.

Si tenemos que guardar archivos que contengan este tipo de información, primero tendremos que almacenar el tipo de cada objeto y después los datos que definen el estado actual del mismo. Cuando recuperemos después esta información desde el archivo, tendremos que:

· Leer el tipo del objeto.
· Crear un objeto en blanco de ese tipo.
· Rellenarlo con los datos almacenados en el archivo.

Esto es posible hacerlo “artesanalmente”, y así es como se hacía. Pero ya no es necesario. Sun Microsystems desarrolló un potente mecanismo que permite efectuar esta operación con mucho menos esfuerzo. Como verá muy pronto, este mecanismo, llamado **serialización de objetos**, casi automatiza por completo lo que antes resultaba ser un proceso muy tedioso.

Almacenar objetos de distinto tipo

Para guardar un objeto, primero tiene que abrir un objeto ObjectOutputStream:

```
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("Empleados.dat"));
```

Ahora, para efectuar el almacenamiento, basta con usar el método writeObject de la clase ObjectOutputStream:

```
Empleados tomas = new Empleados(… Datos de Tomas …);
Gerentes bill = new Gerentes(… Datos de Bill …);
out.writeObject(tomas);
out.writeObject(bill);
```

Para leer los objetos, primero se debe obtener un objeto ObjectlnputStream:

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("Empleados.dat"));
```

Después, los objetos se recuperan en el mismo orden en el que fueron escritos mediante el método readObject:

```
Empleados e1 = (Empleados)in.readObject();
Empleados e2 = (Empleados)in.readObject();
```

Cuando se recuperan objetos, es muy importante saber cuántos de ellos se han guardado, su orden y sus tipos. Cada llamada a readObject lee otro objeto de tipo Object. Por tanto, tendrá que moldearlo al tipo que usted necesite.

Si no necesita el tipo exacto, o no se acuerda de él, puede moldearlo a cualquier superclase o, incluso, dejarlo como Object. Por ejemplo, e2 es una variable Empleados incluso aunque haga referencia a un objeto Gerentes. Si necesita obtener de forma dinámica el tipo de un objeto, puede usar el método getClass usado un poco mas adelante en este apunte.

Con los métodos writeObject/readObject, sólo es posible leer objetos, no números. Para efectuar estas operaciones, debe usar otros métodos como writeInt/readInt o writeDouble/readDouble (las clases de flujo de tipo objeto implementan las interfaces DataInput/DataOutput). Desde luego, los números que se encuentren dentro de objetos (como el campo salario de un objeto Empleados) se guardan y recuperan de manera automática. Recuerde que, en Java, las cadenas y los arrays son objetos y que, por consiguiente, pueden ser manipulados con los métodos writeObject/ readObject.

Sin embargo, existe un agregado que es necesario hacer en cualquier clase que deba trabajar con flujos de tipo objeto. Dicha clase debe implementar la interfaz Serializable:

```
class Empleados implements Serializable { . . . }
```

La interfaz Serializable **no tiene métodos**, por lo que no es necesario cambiar la clase de ninguna forma.

Hagamos un primer ejemplo; en él unicamente pretendemos grabar un objeto, luego modificarlo en memoria y para terminar **recuperarlo** (Leerlo desde disco) demostrando que hemos vuelto a sus valores iniciales. Para ello utilizaremos la clase **ObjItems,** una clase de prueba cuyo objetivo es la grabación/lectura de objetos, su único atributo será un objeto de la clase ArrItems, y sus métodos permiten un tratamiento demostrativo del objetivo propuesto.

Para entender en forma simple lo que hacemos, siga el main(). Allí verá que primero instanciamos un array de 10 items desordenado, lo grabamos, luego lo modificamos ordenandolo por código y finalmente deshacemos esta modificación volviendo los datos a lo que tenemos en el archivo de objetos. El capture de la ejecución muestra estos pasos y el detalle está en los metodos de **class ObjItems.**

```
import java.io.*;
class ObjItems implements Serializable{
   private ArrItems arrItems;
   public ObjItems(int tam, char tipo){
      arrItems = new ArrItems(tam,tipo);
   }
   public void demoGrab() throws IOException{
      FileOutputStream arch=new FileOutputStream("backUp.tmp");
      ObjectOutputStream objOut=new ObjectOutputStream(arch);
      if(objOut!=null){
         objOut.writeObject(arrItems);
         objOut.close();
         System.out.println("Objeto arrItems grabado en backUp.tmp");
      }else System.out.println("Objeto objOut no instanciado (???)");
  }
  public void demoMod(){
     for(int i=0;i<arrItems.talle;i++){
        arrItems.item[i].setCodigo(i);  arrItems.item[i].setValor((float)i);}
     System.out.println(arrItems);
  }
  public void demoLect() throws IOException{
     FileInputStream arch=new FileInputStream("backUp.tmp");
     ObjectInputStream objIn=new ObjectInputStream(arch);
```

```
        if(objIn!=null)
           try{System.out.println("Objetos items leidos");
              while(true)  arrItems = (ArrItems)objIn.readObject();
           }catch (EOFException eof){
                 objIn.close();
                 System.out.println("Lectura terminada");
                 System.out.println(arrItems);
           }catch (ClassNotFoundException nfe){
                 System.out.println("ClassNotFoundException");
           }finally {System.out.println("Demo finished !!!");}
   }
   public static void main(String[] args) throws IOException {
                 char x;  ObjItems grbIt = new ObjItems(10,'R');
                 System.out.println("ArrItems instanciado, <Enter>");
                 x = In.readChar();
                 grbIt.demoGrab();
                 System.out.println("ArrItems backUpeado, <Enter>");
                 x = In.readChar();
                 grbIt.demoMod();
                 System.out.println("ArrItems modificado, <Enter>");
                 x = In.readChar();
                 grbIt.demoLect();
                 System.out.println("ArrItems restaurado...");
              }
}        //        class ObjItems
```

```
Forte(tm) for Java(tm), release 2.0, Communit
Output Window
 Primeros 10 de 10
 elementos Item
2 - 3.3316512
6 - 8.409609
0 - 7.3026004
6 - 4.2417564
2 - 7.3105755
0 - 1.2778374
2 - 4.2153373
2 - 6.079938
1 - 5.512898
5 - 4.9891734
ArrItems instanciado, <Enter>
Objeto arrItems grabado en backUp.tmp
ArrItems backUpeado, <Enter>
 Primeros 10 de 10
 elementos Item
0 - 0.0
1 - 1.0
2 - 2.0
3 - 3.0
4 - 4.0
5 - 5.0
6 - 6.0
7 - 7.0
8 - 8.0
9 - 9.0
```

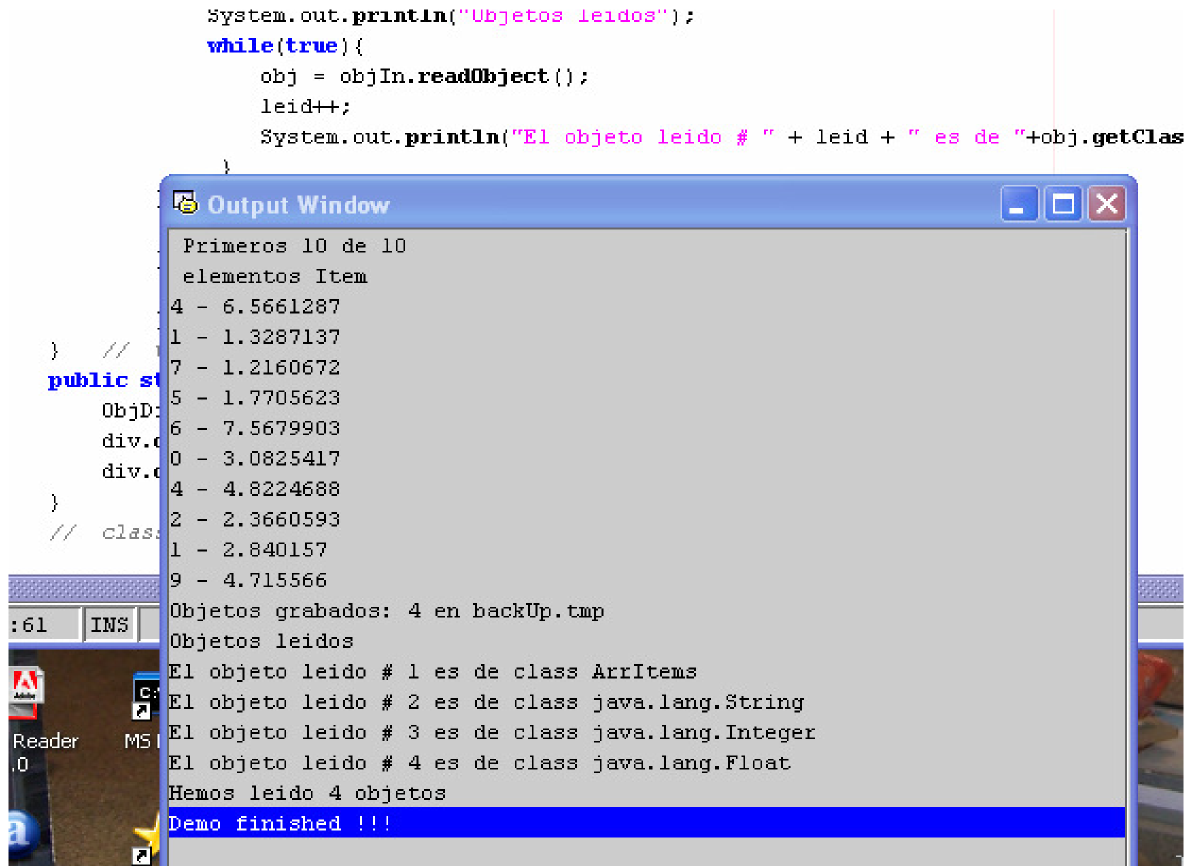
```
ArrItems modificado, <Enter>
Objetos items leidos
Lectura terminada
 Primeros 10 de 10
 elementos Item
2 - 3.3316512
6 - 8.409609
0 - 7.3026004
6 - 4.2417564
2 - 7.3105755
0 - 1.2778374
2 - 4.2153373
2 - 6.079938
1 - 5.512898
5 - 4.9891734

Demo finished !!!
ArrItems restaurado...
```

Vemos que el objeto arrItems contiene sus valores originales. Eso significa que lo hemos grabado y leido con éxito. (En realidad grabamos y leemos un único objeto ArrItems, constituido por objetos Item, del mismo tipo).

Vamos a un segundo ejemplo. Tratando objetos diversos: grabando, leyendo, reconociendo...
En este ejemplo Ud verá que grabamos objetos de distintos tipos, absolutamente diversos. Luego los leemos en una rutina totalmente genérica. Como no deseamos tratarlo de una manera específica los dejamos nomás

en la clase Object. Y como sólo nos interesa saber su clase , pues usamos el método getClass() de la clase Object.

```
import java.io.Serializable;
import java.io.*;
class ObjDiv implements Serializable{
   private ArrItems arrItems;
   private String cadena = "Lo que es verdad a la luz de la lampara...";
   private Integer entero;
   private Float   decimal;
   private int grab =0, leid =0;
   public ObjDiv(int tam, char tipo){
      arrItems = new ArrItems(tam,tipo);
      entero   = new Integer(10);
      decimal  = new Float(10.55);
   }
   public void demoGrab() throws IOException{
      FileOutputStream arch=new FileOutputStream("backUp.tmp");
      ObjectOutputStream objOut=new ObjectOutputStream(arch);
      if( objOut!=null){
         objOut.writeObject(arrItems);   grab++; objOut.writeObject(cadena);     grab++;
         objOut.writeObject(entero);      grab++; objOut.writeObject(decimal);    grab++;
         objOut.close();
         System.out.println("Objetos grabados: " + grab + " en backUp.tmp");
      }else System.out.println("Objeto objOut no instanciado (???)");
   }
   public void demoLect() throws IOException{
      FileInputStream arch=new FileInputStream("backUp.tmp");
      ObjectInputStream objIn=new ObjectInputStream(arch);
      Object obj;
      if( objIn!=null)
         try{
            System.out.println("Objetos items leidos");
            while(true){
               obj = objIn.readObject();
               leid++;
               System.out.println("El objeto leido # " + leid + " es de "+obj.getClass());
            }
         }catch (EOFException eot){ objIn.close();
            System.out.println("Hemos leido " + leid +" objetos");
         }catch (ClassNotFoundException nfe){
            System.out.println("Class Not FoundException ");
         } finally {System.out.println("Demo finished !!! ");}
   }  //  void demoLect()
   public static void main(String[] args) throws IOException {
      ObjDiv div = new ObjDiv(10,'R');
      div.demoGrab();
      div.demoLect();
   }
}  //  class ObjDiv
```



Y un último ejemplo, donde ademas incorporamos un tratamiento polimórfico a los objetos.

Tratando objetos diversos: grabando, leyendo, reconociendo...

```
import java.io.Serializable;
import java.io.*;
class Madre implements Serializable{
   protected String nombre;
   public Madre(String nome){nombre = nome;}
   public String toString(){return nombre;}
}
class Hija extends Madre implements Serializable{
   protected int edad;
   public Hija(String nome, int age){
      super(nome);
      edad = age;
   }
   public String toString(){
      return nombre+", "+edad;
   }
}
class Nieta extends Hija implements Serializable{
   protected String ojos;
   public Nieta(String nome, int age, String eyes){
      super(nome, age);
      ojos = eyes;
   }
   public String toString(){
      return nombre+", "+edad+", "+ojos;
   }
}
```

```
class ObjDivPol implements Serializable{
    private Madre persona[];
    private int grab =0, leid =0;
    public ObjDivPol(){
        persona = new Madre[5];
        persona[0] = new Madre("Dercilia");
        persona[1] = new Hija("Ines", 25);
        persona[2] = new Nieta("Bochi",6,"celestes");
        persona[3] = new Nieta("Maria",5,"negros");
        persona[4] = new Hija("Flopy",8);
   }
   public void demoGrab() throws IOException{
        FileOutputStream arch=new FileOutputStream("familia.tmp");
        ObjectOutputStream objOut=new ObjectOutputStream(arch);
        int i, ind;
        if( objOut!=null){
            for(i= 0;i<10;i++){
                ind = (int)(5*Math.random());
                objOut.writeObject(persona[ind]);
                grab++;
            }
            objOut.close();
            System.out.println("Objetos grabados: " + grab + " en familia.tmp");
        }else System.out.println("Objeto objOut no instanciado (???)");
    }
        public void demoLect() throws IOException{
        FileInputStream arch=new FileInputStream("familia.tmp");
        ObjectInputStream objIn=new ObjectInputStream(arch);
        Object obj;
        if( objIn!=null)
            try{
                System.out.println("Objetos leidos");
                while(true){
                    obj = objIn.readObject();
                    leid++;
                    System.out.println("Leido # " + leid + " es "+obj.getClass());
                    System.out.println(obj);
                 }
            }catch (EOFException eot){ objIn.close();
                System.out.println("Hemos leido " + leid +" personas");
            }catch (ClassNotFoundException nfe){
                System.out.println("Class Not FoundException ");
            } finally {System.out.println("Demo finished !!! ");}
    }   //  void demoLect()
    public static void main(String[] args) throws IOException {
        ObjDivPol divPol = new ObjDivPol();
        divPol.demoGrab();
        divPol.demoLect();
    }
}   //  class ObjDivPol
```

```
Output Window
Objetos grabados: 10 en familia.tmp
Objetos leidos
Leido # 1 es class Hija: Flopy, 8
Leido # 2 es class Hija: Ines, 25
Leido # 3 es class Hija: Flopy, 8
Leido # 4 es class Nieta: Bochi, 6, celestes
Leido # 5 es class Nieta: Maria, 5, negros
Leido # 6 es class Nieta: Maria, 5, negros
Leido # 7 es class Hija: Flopy, 8
Leido # 8 es class Hija: Flopy, 8
Leido # 9 es class Hija: Ines, 25
Leido # 10 es class Hija: Flopy, 8
Hemos leido 10 personas
Demo finished !!!
```

A continuación presento un trabajo de un ingeniero, integrante de la catedra, solucionando un problema de la grabacion al final (Append) de archivos de objetos(serializados). Vamos a usar su propio mail a modo de introducción.

Estimados, buenos días

El día de ayer envié a Tymo y Karina un proyecto para el tema web, de tal manera que en vez de generar todo en memoria, genere archivos, para completar el ciclo con algo de persistencia sin entrar en el modelo relacional, el inconveniente lo encontré cuando quise usar serialización con append, y choque con que no se puede y opte por Binarios como una forma más rápida y fácil. Sin embargo, continuando con el stand by del proyecto en el que trabajo diariamente, me encontré con el tiempo para renegar y poder Serializar la grabación de los archivos en vez de usar binarios, y lo he logrado

Básicamente, investigando, varios foristas indicaron que la mejor manera esta crear clases NoHeaderObject (para grabación y lectura) sobrescribiendo los métodos de leer y escribir los header para que no hagan nada, y de hecho tienen razón, ya que de esa manera se puede realizar acciones append a los archivos bajo serialización.

Lo que hice, es crear un paquete io en mi proyecto con esas clases y listo, de esta manera tenemos una forma más sencilla de simular el modelo de persistencia, sin entrar en un modelo relacional y darle un giro de rosca al tema web, para que de esa manera el alumno tenga la visión completa de lo que es programar en un entorno más profesional

8 **TratandoArchivosBinarios:** *posee grabación de archivos binarios*
8 **AppendandoEnArchivosSerializados:** *tiene las modificaciones realizando append y serialización con las clases nuevas*

Esperando sus comentarios y observaciones como siempre, saludos cordiales

German Romani
Ing en Sistemas de Información

Ambos proyectos han sido desarrollados para el entorno Web que vemos en la asignatura PPR en segundo año. Su contenido (Interfaces graficas usando JSP(Java Server Pages), la arquitectura MVC(Modelo Vista Controlador), etc hace que sea imposible desarrollarlos en este momento. Pero como el tema puede interesar a alumnos que necesiten trabajar en esta modalidad, subimos ambos al sitio Web de la catedra: http://labsys.frc.utn.edu.ar, ir a Sitios de las Catedras, Algoritmos y Estructuras de Datos 2011, Unidad IV, Java.

Almacenamiento en bases de Datos

Introducción.

- Los archivos han sido la solución para la persistencia de datos por muchos años.
- Aun hoy se usan, pero muy limitadamente:
    - Archivos de configuración
    - Archivos tipo texto para migrar datos de una aplicación a otra
    - Los datos que usan y mantienen las aplicaciones actuales están en Bases de Datos.

Definición

- Una base de datos es una colección de datos clasificados y estructurados
    - Son guardados en archivos llamados tablas
    - Es referenciado como si fuera un único archivo.

Creación y manipulación

- Existen varios sistemas administradores de bases de datos (DBMSs)
- Propietarios: Access, SQL Server, Oracle y DB2
- Libre distribución: MySQL y PostgreSQL.
- Usaremos MySQL, un gestor de bases de datos que cubre perfectamente lo que pretendemos enseñar
- Los datos de una base de datos relacional se almacenan en tablas
- Las tablas se relacionan (lógicamente)entre sí utilizando campos clave comunes.
- Cada tabla dispone los datos *en filas* y *columnas.*
- Por ejemplo, piense en el listín de teléfonos.

Los datos relativos a un teléfono

- (nombre, dirección, teléfono, etc.) son *columnas* que agrupamos en una *fila.*
- El conjunto de todas las *filas* de todos los teléfonos forman una tabla de la base de datos.

Nombre	*Dirección*	*Teléfono*
Aguado Rodríguez, Jesús	Las Ramblas 3, Barcelona	932345678
Cuesta Suñer, Ana María	Mayor 22, Madrid	918765432
...	. ..	...

- Una tabla es una colección de datos presentada en forma de una matriz bidimensional
- Las filas reciben también el nombre de *tuplas* o *registros* y las columnas de *campos.*

Operaciones

- como insertar, recuperar, modificar y eliminar datos,
- añadir nuevas tablas o eliminarlas.
- Estas operaciones se expresan en un lenguaje denominado SQL.

SQL (System Query Language)

- SQL es el lenguaje estándar para interactuar con bases de datos relacionales para todos los sistemas administradores de bases de datos actuales.
- Las unidades básicas son *tablas, columnas* y *filas.*

La tabla

- proporciona una forma simple de relacionar sus datos
  - una columna representa un dato presente en la tabla
  - una fila representa un registro o entrada de la tabla.

Crear una base de datos

CREATE DATABASE **<Nombre base de datos>**

Eliminar **una base de datos**
DROP DATABASE **<Nombre base de datos>**

Crear una tabla

- SQL proporciona la sentencia CREATE TABLE.
- Esta sentencia especifica
  - el nombre de la tabla,
  - los nombres y tipos de las columnas de la tabla
  - las claves primaria y externa de esa tabla (externa o foránea, es primaria de otra tabla).

Sintaxis
CREATE TABLE <tabla><columna 1> [,<columna 2>] ...)

Sintaxis de columna

<columna n> <tipo de dato> [DEFAULT <expresión>] [<constante 1> [<constante 2>] ...]

Algunos de los tipos de datos más utilizados son los siguientes:

Tipo SQL	*Tipo Java*
BOOLEAN	boolean
INTEGER	int
REAL	float
FLOAT	double
CHAR	String
VARCHAR	String
BINARY	byte[]
DATE	Java.sql.Date

- La cláusula **DEFAULT** permite especificar un valor por omisión para la columna
  - opcionalmente, para indicar la forma o característica de cada columna,
  - se pueden utilizar las constantes:
    - NOT NULL (no se permiten valores nulos),

  - NULL,
  - UNIQUE
  - PRIMARY KEY.

- La cláusula **PRIMARY KEY** se utiliza para definir la columna como clave principal de la tabla.
  - No puede contener valores nulos ni duplicados; (dos filas no pueden tener el mismo valor en esa columna.
  - Una tabla puede contener una única restricción PRIMARY KEY.

- La cláusula **UNIQUE** indica que la columna no permite valores duplicados; (dos filas no pueden tener el mismo valor en esa columna).
  - Una tabla puede contener varias restricciones UNIQUE.
  - Se suele emplear para que el propio SQL compruebe que no se añaden valores que ya existen.

Ejemplo

```
CREATE TABLE alumnos(
     id_alumno  INTEGER PRIMARY KEY,
     apellidos  VARCHAR(24) NOT NULL,
     nombre     VARCHAR(18) NOT NULL,
     curso            INTEGER NOT NULL,
     titulacion       INTEGER NOT NULL
)
```

La diferencia entre los tipos *CHAR (n)* y *VARCHAR (n)* es que en el primer campo se rellena con espacios hasta *n* caracteres (longitud fija) y en el segundo no (longitud variable).

Agregar datos (filas) a una tabla

- SQL proporciona la sentencia INSERT.
- agrega una o más filas nuevas a una tabla.

Su **sintaxis** simplificada

```
INSERT [INTO] <tabla> [(<columna 1>[,<columna 2>] ... )]
     VALUES (<expresión 1>[,<expresión 2>] ... ) ....

INSERT [INTO] ... SELECT ... FROM ...
```

tabla es el nombre de la tabla en la que se desean insertar las filas,

- seguido por una lista con los nombres de las columnas que van a recibir los datos
- especificados por la lista de valores que siguen a la cláusula VALUES.
- Las columnas no especificadas en la lista reciben el valor NULL, si lo permiten,
- o el valor predeterminado, si se especificó.
- Si todas las columnas reciben datos, se puede omitir la lista con los nombres de las columnas.

Con respecto al segundo formato, un poco más adelante …

Ejemplo

```
INSERT INTO alumnos
   VALUES (324555,'Aguirre Soriano','Leticia',4,3)
```

Modificar datos de una tabla

SQL proporciona la sentencia UPDATE.

- Esta sentencia puede cambiar los valores de
  - filas individuales,
  - grupos de filas
  - todas las filas de una tabla.

Su sintaxis

```
UPDATE <tabla>
     SET <columna 1 = «expresión 1> | NULL)
     [,<columna 2 = «expresión 2> | NULL)]...
     WHERE <condición de búsqueda>
```

La **cláusula SET** contiene

- lista separada por comas de las columnas que deben actualizarse
- el nuevo valor de cada columna.
- El valor suministrado por las expresiones incluye
  - constantes,
  - valores seleccionados de una columna de otra tabla,
  - valores calculados por una expresión compleja.

la **cláusula WHERE** especifica la condición de búsqueda
  - que define la fila de la tabla
  - cuyas columnas se desean modificar.

El **ejemplo** modifica en la tabla **alumnos** los apellidos del alumno que tiene la clave especificada:

```
UPDATE alumnos
    SET apellidos='DIAZ GAGO'
    WHERE id_alumno=5142381
```

Borrar registros de una tabla

SQL proporciona la sentencia DELETE.

- Esta sentencia quita una o varias filas de una tabla.

Su sintaxis

```
DELETE FROM <tabla> WHERE <condición de búsqueda>
```

- Se eliminan todas las filas que satisfacenla cláusula WHERE.
- Si no se especifica una cláusula WHERE, se eliminan todas las filas de la tabla.
- (la tabla sigue permaneciendo en la base de datos)
- si se quiere quitar la tabla de la base de datos

```
DROP TABLE <tabla>
```

Seleccionar datos de una tabla

SQL proporciona la sentencia SELECT.

- Las cláusulas principales se pueden resumir:

```
SELECT [ALL | DISTINCT] <lista de selección>
  FROM <tablas>
  WHERE <condiciones de selección>
  [ORDER BY <columna 1> [ASC|DESC][, <columna 2> [ASC|DESC]] ... ]
```

- Estas cláusulas deben especificarse en el orden indicado.
- **lista de selección** describe las columnas del conjunto de resultados.
  - Es una lista de expresiones separadas por comas.
  - Cada una suele ser
    - una referencia a una columna de la tabla de la que provienen los datos,
    - puede ser cualquier otra expresión.
    - Si usamos * en una **lista de selección** se devolverán todas las columnas de la tabla de origen.

- DISTINCT elimina las repeticiones del conjunto de resultados
- ALL especifica que pueden aparecer filas duplicadas (default)
- FROM especifica la lista de tablas de donde se recuperan los datos
- WHERE describe un filtro
  - que define las condiciones que debe cumplir cada fila
  - de las tablas de origen para satisfacer los requisitos
  - de la instrucción SELECT.
    (Sólo filas que cumplen condiciones son seleccionadas)
- ORDER BY define el orden de las filas del conjunto de resultados
  - ASC,secuencia ascendente
  - DESC, secuencia descendente.

Ejemplos

SELECT * FROM alumnos
// lista tabla **alumnos** completa:

SELECT * FROM alumnos ORDER BY apellidos
// idem ordenadas ascendente por apellido

SQL permite utilizar los operadores

- <, <=, >, >=, <>,
- AND, OR, NOT, IS NULL, LIKE, BETWEEN, IN, ALL, ANY, etc. El ejemplo siguiente

SELECT * FROM alumnos WHERE curso <= 3 AND titulacion = 5
// Recupera los alumnos de la tabla *alumnos* que estén en los cursos **1**,**2** y 3 de la titulación 5

SELECT * FROM alumnos WHERE curso IN(4,5) AND titulacion = 7
// Lista todos los alumnos de la tabla *alumnos* que estén matriculados en los cursos 4 y 5 de la titulación 7:

Podríamos seguir con innúmeros ejemplos.
Pero es mejor trabajar con un programa real

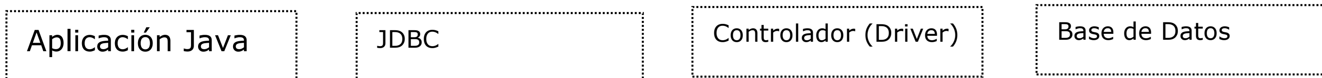
- ver como las cosas efectivamente suceden.

- Para ello debemos conocer como se hace para acceder a una base de datos desde Java.

ACCESO A UNA BASE DE DATOS CON JDBC

JDBC *(Java DataBase Connectivity* - conexión con bases de datos desde Java)

- es una API *(Application Programming Interface* - interfaz de programación de aplicaciones)
    - proporcionada por un conjunto de clases,
    - para ejecutar instrucciones SQL para manipular y gestionar bases de datos relacionales.
    - Para que una aplicación Java pueda hacer operaciones sobre una base de datos, previamente tiene que establecer una conexión con la misma.
        - Esta conexión se realiza a través de un controlador **(driver)**
        - La función del controlador es traducir los mensajes propietarios de bajo nivel del sistema base de datos a mensajes de bajo nivel de la API JDBC y viceversa.



La API JDBC se proporciona en dos paquetes: **java.sql y javax.sql.**

java.sql: Forma parte de J2SE, contiene las interfaces y clases Java fundamentales de JDBC

- **Driver**. Permite conectarse a una base de datos. Cada sistema administrador de base de datos requiere un controlador *(driver)* específico.

- **DriverManager**. Permite gestionar todos los controladores instalados en la máquina virtual de Java.

- **DriverPropertyInfo**. Proporciona diversa información acerca de un controlador.

- **Connection**. Representa una conexión con una base de datos. Una aplicación puede tener abiertas múltiples conexiones con varias bases de datos.

- **DatabaseMetaData**. Proporciona información acerca de una base de datos; por ejemplo, nombre de la base, nombre del controlador, número máximo de conexiones disponibles, etc.

- **Statement**. Permite ejecutar sentencias SQL sin parámetros.

- **PreparedStatement**. Permite ejecutar sentencias SQL con parámetros de entrada.

- **CallableStatement**. Permite ejecutar sentencias SQL con parámetros de entrada y salida (procedimientos almacenados).

- **ResultSet**. Conjunto de resultados. Contiene las filas obtenidas al ejecutar una sentencia SELECT.

- **ResultSetMetaData**. Permite obtener información sobre un **ResultSet**;

por ejjemplo, el número de columnas, sus nombres, sus tipos, etc.

De estas clases, cualquier aplicación Java utilizará casi siempre estas cuatro:

- **DriverManager**, utilizada para crear
- un objeto **Connection**, objeto que será utilizado para crear
- un objeto **Statement**, que a su vez será utilizado para crear
- un objeto **ResultSet**.

javax.sql: también se suministra con J2SE.

- Es una extensión de **java.sql**
- incluye las clases que interactúan con JNDI
- otras que gestionan conjuntos de conexiones con una misma base de datos

Controladores *(drivers)*

- Proporcionados por fabricantes particulares
- Un **controlador JDBC** es una clase que implementa la interfaz que proporciona los métodos necesarios para acceder a los datos de la base de datos.
- Hay distintos tipos de controladores JDBC.
- Nosotros vamos a utilizar uno escrito en Java
- que permite la comunicación directa con la base de datos
- Este tipo de controlador es específico para una base de datos;
  - si cambiamos de motor de base de datos,
  - tenemos que cambiar también de controlador
  - utilizar otro que haya sido escrito para el nuevo motor utilizado.

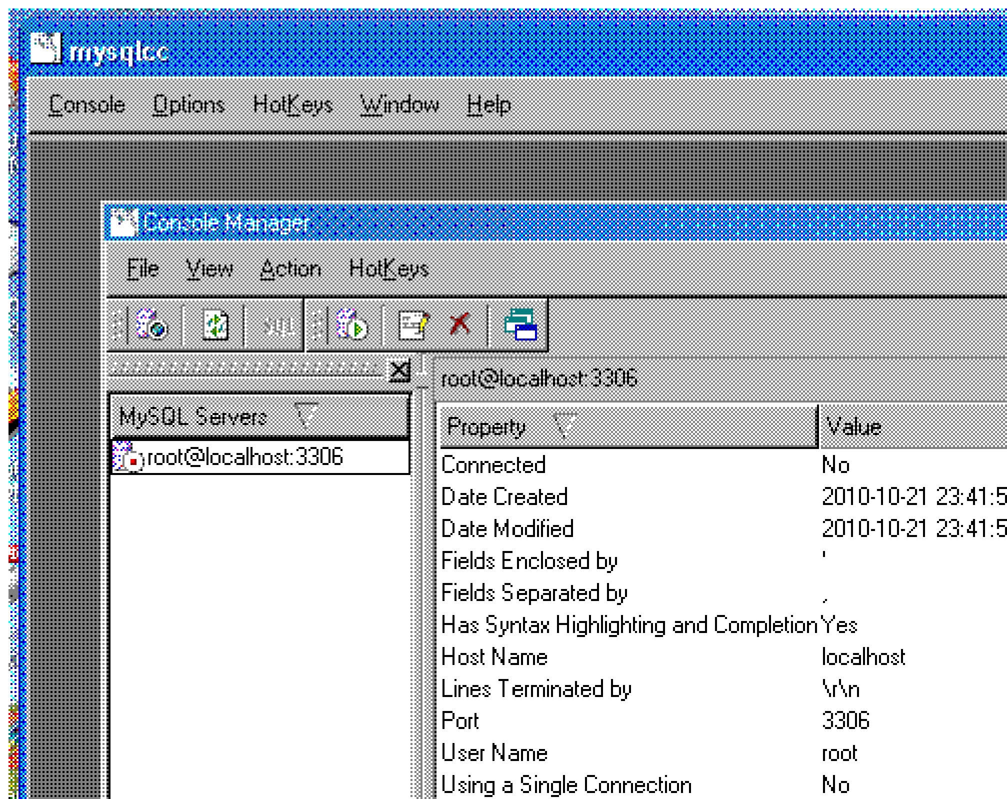
Vamos a trabajar en un contexto Universidad.

- Esto es, la universidad oferta titulaciones
- se nutre de alumnos.
- Estos se matriculan en una u otra titulación
- cursan un número determinado de asignaturas
- hasta lograr aprobarlas a todas (las obligatorias)
- que les proporcionará el título académico correspondiente.
- Para hacer un seguimiento de los alumnos matriculados,
  - necesitaremos registrar una serie de datos,
    - de los alumnos
    - de las asignaturas cursadas.

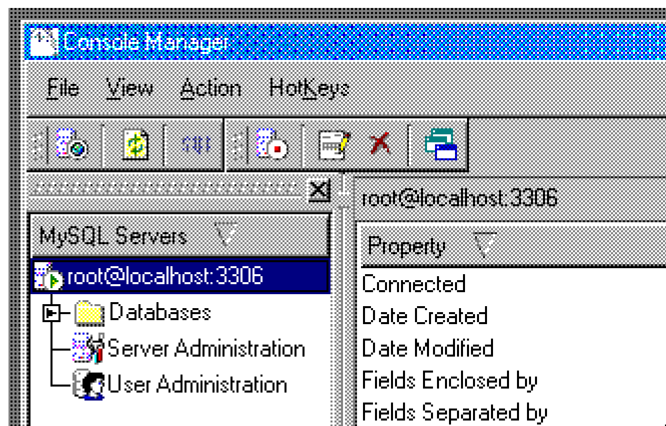
Creación de la base de datos

- tarea específica del DBMS
- en nuestro caso de *MySQLCC*.
- vamos a crear la base de datos que ya detallamos: **bd_alumnos**.
- Tanto la base como sus tablas las generamos con el DBMS MySQLCC (Previamente instalado por personal de laboratorio)

Usando MySQLCC



Posicionamos sobre root@localhost:3306 <Boton derecho> <Conectar>



Debajo de Action (no se ve bien en filmina) tenemos SQL

- Posicionamos cursor, microhelp dice Query (ctrl+Q) (Formular sentencias SQL) <SQL>

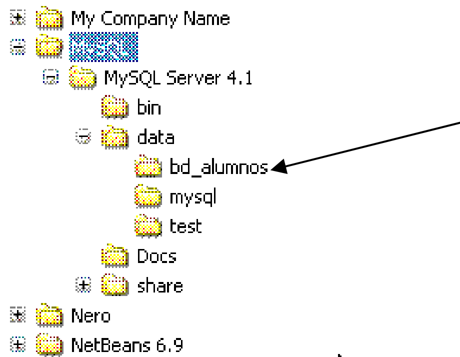
Tenemos la ventana de sentencias u órdenes SQL.

Comenzamos por crear La base de datos:

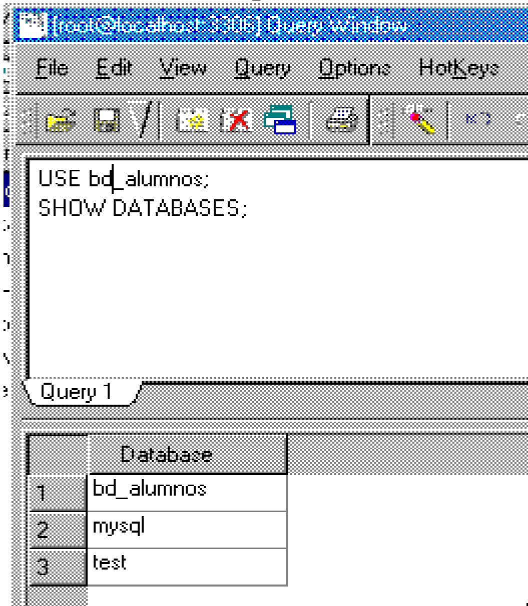
bd_alumnos

Control para ejecutar Query

Vemos en el cuadro de mensajes: "Database created"
Donde la crea?



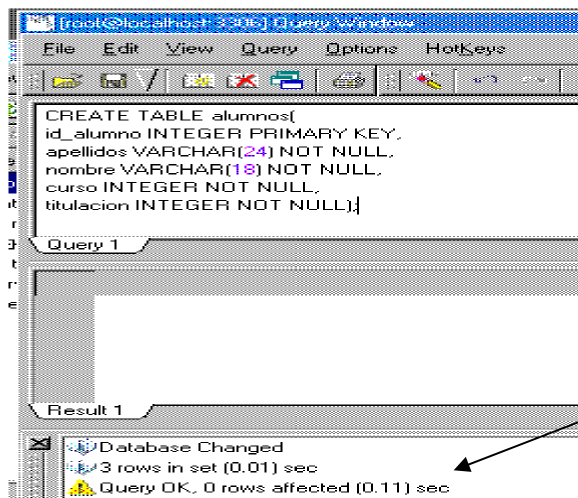
Cuantas bases ya tenemos?



Nos posicionamos en bd_alumnos; (USE)
Pedimos saber que bases existen (SHOW)

A continuación vamos a crear 3 tablas:

- alumnos
- asignaturas
- alumnos_asignaturas



Todo bien.

[root@localhost:3306] Query Window

File Edit View Query Options HotKeys

```
CREATE TABLE asignaturas(
id_asignatura INTEGER PRIMARY KEY,
tipo VARCHAR(2) NOT NULL,
nombre VARCHAR(60) NOT NULL,
creditos FLOAT NOT NULL);
```

```
CREATE TABLE alumnos_asignaturas(
id_alumno INTEGER NOT NULL,
id_asignatura INTEGER NOT NULL,
cursada CHAR(1) NOT NULL);
```

Podemos cargar datos
Comenzamos por la tabla alumnos. Su descripción

```
DESCRIBE alumnos;
```

Query 1

	Field	Type	Null	Key	Default
1	id_alumno	int(11)		PRI	0
2	apellidos	varchar(24)			
3	nombre	varchar(18)			
4	curso	int(11)			0
5	titulacion	int(11)			0

```
INSERT INTO alumnos
VALUES(57749,"Acuña","Javier Emiliano", 9,1);
INSERT INTO alumnos
VALUES(53520,"Alarcón","Pablo Ezequiel", 9,1);
INSERT INTO alumnos
VALUES(53520,"Alberti","Huber Fabricio", 9,1);
INSERT INTO alumnos
VALUES(53520,"Amado Ramos","Marco Antonio", 9,1);
```

Query 1

```
Query OK, 1 row affected (0.03) sec
Query OK, 1 row affected (0.00) sec
[root@localhost:3306] ERROR 1062: Duplicate entry '53520' for key 1
[root@localhost:3306] ERROR 1062: Duplicate entry '53520' for key 1
```

Olvidamos corregir los legajos.
MySQLCC no olvidó advertirnos del error
Corregimos…

```
INSERT INTO alumnos
VALUES(58878,"Alberti","Huber Fabricio", 9,1);
INSERT INTO alumnos
VALUES(57572,"Amado Ramos","Marco Antonio de Cleo", 9,1);
INSERT INTO alumnos
VALUES(55709,"Amaya","Francisco Gastón", 9,1);
INSERT INTO alumnos
VALUES(58919,"Andonian","Diego Martin", 9,1);
```

Query 1

```
Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.02) sec
```

Todo bien, veamos que tenemos

```
SELECT id_alumno, apellidos, nombre FROM alumnos;
```

Query 1

	id_alumno	apellidos	nombre
1	53520	Alarcón	Pablo Ezequiel
2	55709	Amaya	Francisco Gastón
3	57572	Amado Ramos	Marco Antonio de C
4	57749	Acuña	Javier Emiliano
5	58878	Alberti	Huber Fabricio
6	58919	Andonian	Diego Martin

Nombre incorrecto

Debemos corregirlo

```
UPDATE alumnos SET nombre = "Marco Antonio"  WHERE id_alumno = 57572;
```

Query 1

Result 1

Query OK, 1 row affected (0.05) sec
Rows matched: 1 Changed: 1 Warnings: 0

```
SELECT * FROM alumnos WHERE id_alumno = 57572;
```

Query 1

	id_alumno	apellidos	nombre	curso	titulacion
1	57572	Amado Ramos	Marco Antonio	9	1

Veamos como quedó

Perfecto

Carguemos algunas asignaturas.

```
INSERT INTO asignaturas(id_asignatura, tipo,nombre,creditos)
        VALUES(1,"TP","Algoritmos y Estructuras de Datos",6.0);
INSERT INTO asignaturas  VALUES(2,"TP","Paradigmas de Programación",7.0);
INSERT INTO asignaturas  VALUES(3,"T ","Matemática Discreta",4.0);
INSERT INTO asignaturas  VALUES(4,"TP","Física I",4.0);
```

Query 1

Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.00) sec
Query OK, 1 row affected (0.00) sec

Carguemos actuación de alumnos en alumnos_asignaturas. Que alumnos tenemos?

```
SELECT id_alumno, apellidos FROM alumnos;
```

Query 1

	id_alumno	apellidos
1	53520	Alarcón
2	55709	Amaya
3	57572	Amado Ramos
4	57749	Acuña
5	58878	Alberti
6	58919	Andonian

Que tenemos sobre ellos? El primer parcial…

Legajo	Apellido y Nombre	
57749	Acuña, Javier Emiliano	2
53520	Alarcón, Pablo Ezequiel	
58878	Alberti, Huber Fabricio	2
57572	Amado Ramos, Marco Antonio	
55709	Amaya, Francisco Gaston	
58919	Andonian, Diego Martin	8

Entonces cargamos lo realizado

- 2 alumnos cursando, "C"
- 1 alumno regulariz. , "R"
- Resto nada

```
INSERT INTO alumnos_asignaturas VALUES(57749,1,"C");
INSERT INTO alumnos_asignaturas VALUES(58878,1,"C");
INSERT INTO alumnos_asignaturas VALUES(58919,1,"R");
INSERT INTO alumnos_asignaturas VALUES(57749,3,"C");
INSERT INTO alumnos_asignaturas VALUES(58878,3,"C");
INSERT INTO alumnos_asignaturas VALUES(58919,3,"R");
```

Suponemos igual actuación en Matemática Discreta.

Veamos que se cargó.

```
SELECT * FROM alumnos_asignaturas;
```

Query 1

	id_alumno	id_asignatura	cursada
1	57749	1	C
2	58878	1	C
3	58919	1	R
4	57749	3	C
5	58878	3	C
6	58919	3	R

Todo OK.
Si quisiéramos saber quienes están cursando AED

```
SELECT * FROM alumnos_asignaturas
    WHERE cursada = "C" and id_asignatura = 1;
```

Query 1

	id_alumno	id_asignatura	cursada
1	57749	1	C
2	58878	1	C

Perfecto. Ahora solo quiero saber cuantos son.

```
SELECT COUNT(*) FROM alumnos_asignaturas
    WHERE cursada = "C" and id_asignatura = 1;
```

Query 1

	COUNT(*)
1	2

Mi curiosidad va en aumento.
Quiero tener bien identificados que alumnos están cursando o ya han regularizado (Suponiendo…)
Necesito sus apellidos, nombre, cursada, id_asignatura.
Debo relacionar 2 tablas

```
SELECT apellidos, nombre, cursada, id_asignatura
FROM `alumnos` alu, `alumnos_asignaturas` asi
where alu.id_alumno = asi.id_alumno
```

Query 1

	apellidos	nombre	cursada	id_asignatura
1	Acuña	Javier Emiliano	C	1
2	Alberti	Huber Fabricio	C	1
3	Andonian	Diego Martin	R	1
4	Acuña	Javier Emiliano	C	3
5	Alberti	Huber Fabricio	C	3
6	Andonian	Diego Martin	R	3

En realidad lo que quería saber es quienes están cursando AED, sin haberla aun regularizado.

Reformulamos el query

```
SELECT apellidos, nombre, cursada, id_asignatura
FROM `alumnos` alu, `alumnos_asignaturas` asi
where alu.id_alumno = asi.id_alumno
and cursada = "C"
and id_asignatura = 1
```

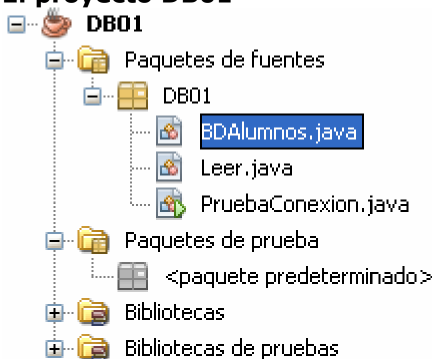
Query 1

	apellidos	nombre	cursada	id_asignatura
1	Acuña	Javier Emiliano	C	1
2	Alberti	Huber Fabricio	C	1

Bueno, ya hemos jugado un poco con SQL usando MySQLCC. Claro, si conocemos el lenguaje SQL lo podemos hacer. El desafio que proponemos es desarrollar una aplicación para que **usuarios que no sepan de SQL** puedan trabajar con una base de datos. Esto lo hacemos en

El proyecto DB01.

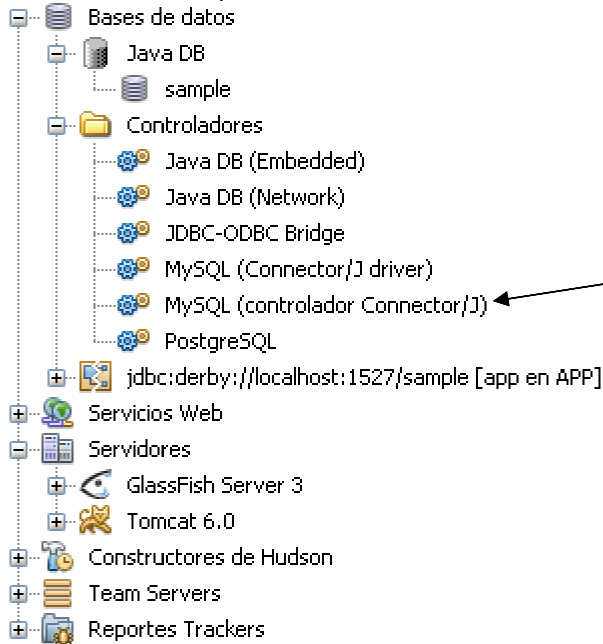
El proyecto DB01



La clase *BDAlumnos* encapsulará la base de datos *bd_alumnos* y mostrará una interfaz pública que permita recuperar datos, insertarlos, borrarlos, etc.,

PruebaConexion, utilizando la interfaz de *BDAlumnos*, mostrará que cualquier usuario sin conocimientos de SQL ni de bases de datos puede trabajar con dicha base

Si abrimos la solapa **Prestaciones**



Vemos que Netbeans viene provisto de un buen número de controladores.

Nosotros hemos agregado este

La aplicación tiene un **main()** en class **PruebaConexion**

```
public class PruebaConexion{
  private static BDAlumnos BD;  //   Objeto de la clase BDAlumnos

  public static void main(String args[])
  {
    int i = 0, opción = 0;
    PruebaConexion objAp = null;
    try{
      objAp = new PruebaConexion();  //   Aquí me conecto a bd_alumnos
      System.out.println("main/try, luego de objAp = new
                                               PruebaConexion()");
      // Opciones del menú
      String[] opciones = { "Datos de la tabla",
                            "Buscar filas en \"alumnos\"",
                            "Insertar fila en \"alumnos\"",
                            "Borrar fila en \"alumnos\"",
                            "Navegar",
                            "Salir."};
      // Nombre de las tablas de la base
      String[] tablas = BD.tablas();
      do
      {
        switch(opción = objAp.menú(opciones, opciones.length))
        {
          case 1:
            i = objAp.menú(tablas, tablas.length);
            BD.mostrarTabla(tablas[i-1]);
            break;
          case 2:
            objAp.buscarFilasEnAlumnos();
            break;
```

```
          case 3:
            objAp.insertarFilaEnAlumnos();
            break;
          case 4:
            objAp.borrarFilaEnAlumnos();
            break;
          case 5:
            i = objAp.menú(tablas, tablas.length);
            BD.obtenerTabla(tablas[i-1]);
            objAp.navegar();
            break;
        }
      }
      while(opción != 6);
    }
    catch(ClassNotFoundException e)
    {
      System.out.println(e.getMessage());
    }
    catch(InstantiationException e)
    {
      System.out.print(e.getMessage());
    }
    catch(IllegalAccessException e)
    {
      System.out.print(e.getMessage());
    }
    catch(java.sql.SQLException e)
    {
      System.out.print(e.getMessage());
    }
    finally  // pase lo que pase cerramos la conexión
    {
      try{BD.cerrarConexion();}
      catch(java.sql.SQLException ignorada) {}
    }
  }
```

```
La ejecución del main:
run:
Conexión realizada con éxito.
Tablas de la base de datos:
alumnos
asignaturas
Alumnos_Asignaturas
____________________________________

    1. Datos de la tabla
    2. Buscar filas en "alumnos"
    3. Insertar fila en "alumnos"
    4. Borrar fila en "alumnos"
    5. Navegar
    6. Listar Alumnos regulares
    7. Salir.
____________________________________
Opción (1 - 7):
```

```
    1. Datos de la tabla
    2. Buscar filas en "alumnos"
    3. Insertar fila en "alumnos"
    4. Borrar fila en "alumnos"
    5. Navegar
    6. Salir.
```

Optamos por
1 ***Datos de la tabla***

Se ejecutara:
BD.mostrarTabla(tablas[i-1]);

```
Opción (1 - 6):1

  public void mostrarTabla(String tabla)
      throws java.sql.SQLException
  {
    cdr = obtenerTabla(tabla);
    while(cdr.next()) mostrarFilaActual();
  }
```

cdr *se define*

```
private  java.sql.ResultSet  cdr;
// conjunto de resultados
```

```
public  java.sql.ResultSet  obtenerTabla
(String tabla)
      throws java.sql.SQLException{
    cdr = sentenciaSQL.executeQuery(
       "SELECT * FROM " + tabla);
    return cdr;
}
```

```
 public void mostrarFilaActual() throws
                     java.sql.SQLException{
   int nColumnas =
          cdr.getMetaData().getColumnCount();
   for (int i = 1; i <= nColumnas; ++i){
     System.out.print(cdr.getString(i) + " ");
   }
   System.out.println();
}
```

La ejecución de **mostrarTabla()**:

```
    1. alumnos
    2. asignaturas
    3. Alumnos_Asignaturas

Opción (1 - 3):2

1 TP Algoritmos y Estructuras de Datos 6
2 TP Paradigmas de Programación 7
3 T Matemática Discreta 4
4 TP Física I 4
```

Estamos optando por datos de las asignaturas, entonces:

- **obtenerTabla**(*Nombre tabla*):Construimos **cdr**: conjunto de filas de la tabla asignaturas
- while(**cdr.next()**): Ejecutamos un ciclo recorriéndolo y listando esas filas
- Volvemos al menú principal

```
________________________________________

    1. Datos de la tabla
    2. Buscar filas en "alumnos"
    3. Insertar fila en "alumnos"
    4. Borrar fila en "alumnos"
    5. Navegar
    6. Salir.
________________________________________


Opción (1 - 6):2
```

Ahora vamos a buscar filas en alumnos

Se ejecutará:
objAp.buscarFilasEnAlumnos()

```
  public void buscarFilasEnAlumnos()
      throws java.sql.SQLException
  {
    System.out.println("\nBuscar:");
    String[] búsquedas = {"Apellidos que empiecen por...",
                          "Apellidos que contengan..." };
    int i = menú(búsquedas, búsquedas.length);

    System.out.print("> ");
    String subcadena = Leer.dato();

    BD.mostrarFilasDeAlumnos(subcadena, i);
  }
```

Su ejecución

```
Buscar:

________________________________________

    1. Apellidos que empiecen por...
    2. Apellidos que contengan...
________________________________________


Opción (1 - 2):1
Opción (1 - 2): 1
> An
```

La ejecución de **MostrarFilasDeAlumnos**(subcadena, i)

```
  public void mostrarFilasDeAlumnos(String subcad, int tipoBusqueda)
      throws java.sql.SQLException
  {
    cdr = buscarFilasEnAlumnos(subcad, tipoBusqueda);
    while(cdr.next()) mostrarFilaActual();
  }


  public java.sql.ResultSet buscarFilasEnAlumnos(String subcad,
      int tipoBúsqueda) throws java.sql.SQLException
  {
    String[] cadena = {"'" + subcad + "%'", "'%" + subcad + "%'"};
    cdr = sentenciaSQL.executeQuery(
        "SELECT * FROM " + "alumnos" +
```

```
           " WHERE apellidos LIKE " + cadena[tipoBúsqueda-1]);
    return cdr;
  }
```

El único apellido iniciado con “An”:

58919 Andonian Diego Martin 9 1

Faltó ver la conexión a la base de datos: En el main()

objAp = new PruebaConexion();

```
public PruebaConexion() throws ClassNotFoundException,
      java.sql.SQLException, InstantiationException,
                             IllegalAccessException{
    try{
        BD = new BDAlumnos();   //   Constructor
    }catch(java.sql.SQLException SQLExc){
        System.out.println("PruebaConexion()"+SQLExc.getMessage());
    }
    // Realizar la conexión con la base de datos BD
    System.out.println("Estoy en PruebaConexion(), luego de BD = new
                                                  BDAlumnos();");

  }

public class BDAlumnos{
  private java.sql.Connection conexión;
  private java.sql.Statement sentenciaSQL;
  private java.sql.ResultSet cdr; // conjunto de resultados

  public BDAlumnos() throws ClassNotFoundException,
      java.sql.SQLException, InstantiationException,
      IllegalAccessException{

    // Cargar el controlador JDBC-ODBC
    String controlador = "sun.jdbc.odbc.JdbcOdbcDriver";
    Class.forName(controlador).newInstance();
    conectar(); // conectar con el origen de datos
  }

  public void conectar() throws java.sql.SQLException{

    String URL_bd = "jdbc:mysql://127.0.0.1:3306/bd_alumnos";
    String usuario = "root";
    String contraseña = "root";
    // Conectar con la BD
    try{
        conexión = java.sql.DriverManager.getConnection(
                            URL_bd, usuario, contraseña);

    }catch(Exception exc){System.out.println("Fracasa getConnection " +
                                 exc.getMessage());
    }

    System.out.println("\nConexión realizada con éxito.\n");
```

```
    // Crear una sentencia SQL
    sentenciaSQL = conexión.createStatement(
                       java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE,
                       java.sql.ResultSet.CONCUR_UPDATABLE);

    // Mostrar las tablas de la base de datos
    System.out.println("Tablas de la base de datos: ");
    String[] tabla = tablas();
    for (int i = 0; i < tabla.length; ++i)
      System.out.println(tabla[i]);
  }

//    Vamos agregar una opción al menú
          case 6: objAp.listarAlumnosRegulares(); break;

  public void listarAlumnosRegulares()throws java.sql.SQLException{
    cdr = obtenerAlumnosRegulares();
    while(cdr.next()) mostrarFilaActual();
  }

  public java.sql.ResultSet obtenerAlumnosRegulares()
                              throws java.sql.SQLException{
    cdr = sentenciaSQL.executeQuery(
        "SELECT alu.id_alumno, apellidos, nombre, asi.cursada,
                                                asi.id_asignatura "
           + "FROM " + "alumnos alu" + ", alumnos_asignaturas asi "
           + "WHERE alu.id_alumno = asi.id_alumno "
           + "AND asi.cursada = 'R'");
        return cdr;
  }
```

```
Opción (1 - 7): 6
58919 Andonian Diego Martin R 1
58919 Andonian Diego Martin R 3
```

// Suficiente!!!