EGISTROS Y ARCHIVOS

Como ya sabemos hay variables simples que pueden almacenar una sola pieza de información y como arreglos pueden almacenar un conjunto de ellas del mismo tipo y al mismo tiempo, estos dos mecanismos pueden manejar una gran variedad de situaciones, pero a menudo se necesita trabajar sobre datos de diversos tipos, en este caso ni variables escalares ni arreglos son adecuados.
Para resolver estos problemas los lenguajes de programación proveen de un tipo de dato especial llamado registros.

Un registro es una variable especial que tiene la capacidad de almacenar datos de diferentes tipos.

Sin embargo, JAVA, usa en su lugar una CLASE.

Este método tiene la ventaja de que además de incluir los campos tradicionales de un registro (en forma de atributos), también puede incorporar una serie de metodos que permiten procesar de manera más fácil los campos o atributos de la clase.

Ejemplo;

```
class alumno{
void alumno(){};
static String nombre= new String();
static int edad;
void inicializar(){
alumno.nombre="pepe el toro";
alumno.edad=18; };
void desplegar(){
System.out.println(alumno.nombre);
System.out.println(alumno.edad);
};
} termina la clase
```

Programa ejemplo;

```
import java.io.*;

class prog33{

public static void main(String[] args) {

alumno.inicializar();

alumno.desplegar();

} // cierra main

} // cierra clase

class alumno{

void alumno(){};

static String nombre= new String();

static int edad;

static void inicializar(){

alumno.nombre="pepe el toro";

alumno.edad=18; };

static void desplegar(){

System.out.println(alumno.nombre);

System.out.println(alumno.edad);

};
} // temina la clase
```

Para indicar a "java" durante cualquier proceso que la variable a utilizar es un campo de una clase, se deberá utilizar el siguiente formato.

nomclase.nombredelcampo

Codigo ejemplo:

En el siguiente código tenemos lo siguiente:
Un método en Java cuya función es escribir un nuevo archivo 1 simple línea de texto. EL método consta de un parámetro que es el nombre del archivo, por ejemplo "archivo.txt". Debe estar incluida la extensión en el nombre, pues no se asigna por defecto.

Veamos el código9

```
import java.io.*;//no olviden importar esta librería al inicio de su programa

 //esto es solo un método, no una clase, el método deberá ser incluido en una <span class="m782206" id="m782206_4">clase java</span> para su uso

public void escribir(String nombreArchivo)



{

File f;

f = new File("nombreArchivo");

//Escritura

try{

FileWriter w = new FileWriter(f);

BufferedWriter bw = new BufferedWriter(w);

PrintWriter wr = new PrintWriter(bw);

wr.write("Esta es una linea de codigo");//escribimos en el archivo

wr.append(" - y aqui continua"); //concatenamos en el archivo sin borrar lo existente

//ahora cerramos los flujos de canales de datos, al cerrarlos el archivo quedará guardado con información escrita

        //de no hacerlo no se escribirá nada en el archivo

wr.close();

bw.close();

}catch(IOException e){};

}
```

Como se puede apreciar, es necesario incluir el código dentro de un "try" y un "catch" para evitar errores.

Será necesario el uso de 4 clases especiales para poder escribir, la clase File, FileWriter, BufferedWriter y PrintWriter, cada una hace lo siguiente:

1. File: esta clase es la escencia de crear un nuevo archivo, si un archivo con el mismo nombre ya existe podríamos sin querer escribir contenido sobre el mismo.

2. FileWriter: es un objeto que tiene como función escribir datos en un archivo.

3. BufferedWriter: objeto que reserva un espacio en memoria donde se guarda la información antes de ser escrita en un archivo.

4. PrintWriter: Es el objeto que utilizamos para escribir directamente sobre el archivo de texto.

Lectura y Escritura de Ficheros en Java

Lectura de un fichero de texto en java

Podemos *abrir un fichero de texto* para leer usando la clase *FileReader*. Esta clase tiene métodos que nos permiten leer caracteres. Sin embargo, suele ser habitual querer las líneas completas, bien porque nos interesa la línea completa, bien para poder analizarla luego y extraer campos de ella. *FileReader* no contiene métodos que nos permitan leer líneas completas, pero sí *BufferedReader*. Afortunadamente, podemos construir un *BufferedReader* a partir del *FileReader* de la siguiente forma:

```
File archivo = new File ("C:\\archivo.txt");

FileReader fr = new FileReader (archivo);

BufferedReader br = new BufferedReader(fr);

...

String linea = br.readLine();
```

La apertura del fichero y su posterior lectura pueden lanzar excepciones que debemos capturar. Por ello, la apertura del fichero y la lectura debe meterse en un bloque *try-catch*.

Además, el fichero hay que cerrarlo cuando terminemos con él, tanto si todo ha ido bien como si ha habido algún error en la lectura después de haberlo abierto. Por ello, se suele poner al*try-catch* un bloque *finally* y dentro de él, el *close()* del fichero.

El siguiente es un código completo con todo lo mencionado.

```
import java.io.*;

class LeeFichero {
   public static void main(String [] arg) {
      File archivo = null;
      FileReader fr = null;
      BufferedReader br = null;

      try {
         // Apertura del fichero y creacion de BufferedReader para poder
         // hacer una lectura comoda (disponer del metodo readLine()).
         archivo = new File ("C:\\archivo.txt");
         fr = new FileReader (archivo);
         br = new BufferedReader(fr);

         // Lectura del fichero
         String linea;
         while((linea=br.readLine())!=null)
            System.out.println(linea);
      }
      catch(Exception e){
         e.printStackTrace();
      }finally{
         // En el finally cerramos el fichero, para asegurarnos
         // que se cierra tanto si todo va bien como si salta
         // una excepcion.
```

```
        try
```

Como opción para leer un fichero de texto línea por línea, podría usarse la clase *Scanner* en vez de el *FileReader* y el *BufferedReader*. Ver el ejemplo del Ejemplo de lectura de un fichero con Scanner

---

Escritura de un fichero de texto en java

El siguiente código **escribe un fichero de texto** desde cero. Pone en él 10 líneas

```
import java.io.*;

public class EscribeFichero{

    public static void main(String[] args)    {

        FileWriter fichero = null;

        PrintWriter pw = null;

        try

        {

            fichero = new FileWriter("c:/prueba.txt");

            pw = new PrintWriter(fichero);

            for (int i = 0; i < 10; i++)

                pw.println("Linea " + i);

    } catch (Exception e) {

            e.printStackTrace();

        } finally {         try {

           // Nuevamente aprovechamos el finally para

           // asegurarnos que se cierra el fichero.
```

```
            if (null != fichero)
               fichero.close();
            } catch (Exception e2) {
               e2.printStackTrace();
            }
         }
    }
}
```

Si queremos **añadir al final de un fichero** ya existente, simplemente debemos poner un flag a true como segundo parámetro del constructor de **FileWriter**.

FileWriter fichero = new FileWriter("c:/prueba.txt",**true**);

Ficheros binarios

Para ficheros binarios se hace exactamente igual, pero en vez de usar los "**Reader**" y los "**Writer**", se usan los "**InputStream**" y los "**OutputStream**". En lugar de los **readLine()** y**println()**, hay que usar los métodos **read()** y **write()** de array de bytes.

El siguiente ejemplo hace una copia binaria de un fichero

```
package chuidiang.ejemplos;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class CopiaFicheros {

    public static void main(String[] args) {
```

```
        copia ("c:/ficheroOrigen.bin", "c:/ficheroDestino.bin");
    }

    public static void copia (String ficheroOriginal, String ficheroCopia)
    {
        try
        {
                        // Se abre el fichero original para lectura
            FileInputStream fileInput = new
FileInputStream(ficheroOriginal);
            BufferedInputStream bufferedInput = new
BufferedInputStream(fileInput);

            // Se abre el fichero donde se hará la copia
            FileOutputStream fileOutput = new FileOutputStream
(ficheroCopia);
            BufferedOutputStream bufferedOutput = new
BufferedOutputStream(fileOutput);

            // Bucle para leer de un fichero y escribir en el otro.
            byte [] array = new byte[1000];
            int leidos = bufferedInput.read(array);
            while (leidos > 0)
            {
                bufferedOutput.write(array,0,leidos);
                leidos=bufferedInput.read(array);
            }

            // Cierre de los ficheros
            bufferedInput.close();
```

```
            bufferedOutput.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Los Buffered*

Si usamos sólo **FileInputStream**, **FileOuputStream**, **FileReader** o **FileWriter**, cada vez que hagamos una lectura o escritura, se hará fisicamente en el disco duro. Si escribimos o leemos pocos caracteres cada vez, el proceso se hace costoso y lento, con muchos accesos a disco duro.

Los **BufferedReader**, **BufferedInputStream**, **BufferedWriter** y **BufferedOutputStream** añaden un buffer intermedio. Cuando leamos o escribamos, esta clase controlará los accesos a disco.

- Si vamos escribiendo, se guardará los datos hasta que tenga basantes datos como para hacer la escritura eficiente.
- Si queremos leer, la clase leerá muchos datos de golpe, aunque sólo nos dé los que hayamos pedido. En las siguientes lecturas nos dará lo que tiene almacenado, hasta que necesite leer otra vez.

Esta forma de trabajar hace los accesos a disco más eficientes y el programa correrá más rápido. La diferencia se notará más cuanto mayor sea el fichero que queremos leer o escribir.

Crea un archivo en java de acceso aleatorio, escribiendo 100 registros vacios en el disco

```
Crea un archivo de acceso aleatorio, escribiendo 100 registros vacíos en el disco.

import java.io.*;

import javax.swing.*;
```

```
public class CrearArchivoAleatorio {


    private static final int NUMERO_REGISTROS = 100;


   // permitir al usuario seleccionar el archivo a abrir

   private void crearArchivo()

   {

      // mostrar cuadro de diálogo para que el usuario pueda seleccionar
el archivo

      JFileChooser selectorArchivo = new JFileChooser();

      selectorArchivo.setFileSelectionMode( JFileChooser.FILES_ONLY );


      int resultado = selectorArchivo.showSaveDialog( null );


      // si el usuario hizo clic en el botón Cancelar del cuadro de
diálogo, regresar

      if ( resultado == JFileChooser.CANCEL_OPTION )

         return;


      // obtener el archivo seleccionado

      File nombreArchivo = selectorArchivo.getSelectedFile();


      // mostrar error si el nombre del archivo es inválido
```

```
      if ( nombreArchivo == null || nombreArchivo.getName().equals( "" )
)

         JOptionPane.showMessageDialog( null, "Nombre de archivo
incorrecto",

            "Nombre de archivo incorrecto", JOptionPane.ERROR_MESSAGE );


      else {


         // abrir el archivo

         try {

            RandomAccessFile archivo =

               new RandomAccessFile( nombreArchivo, "rw" );


            RegistroCuentasAccesoAleatorio registroEnBlanco =

               new RegistroCuentasAccesoAleatorio();


            // escribir 100 registros en blanco

            for ( int cuenta = 0; cuenta < NUMERO_REGISTROS; cuenta++ )

               registroEnBlanco.escribir( archivo );


            archivo.close(); // cerrar el archivo


            // mostrar mensaje indicando que el archivo se creó

            JOptionPane.showMessageDialog( null, "Se creó el archivo " +

               nombreArchivo, "Estado", JOptionPane.INFORMATION_MESSAGE
);


            System.exit( 0 );  // terminar el programa
```

```
         } // fin del bloque try


         // procesar excepciones durante operaciones de apertura,
escritura o cierre del archivo
         catch ( IOException excepcionES ) {
            JOptionPane.showMessageDialog( null, "Error al procesar el
archivo",
               "Error al procesar el archivo", JOptionPane.ERROR_MESSAGE
);


            System.exit( 1 );
         }


      } // fin de instrucción else


   } // fin del método crearArchivo


   public static void main( String args[] )
   {
      CrearArchivoAleatorio aplicacion = new CrearArchivoAleatorio();
      aplicacion.crearArchivo();
   }


} // fin de la clase CrearArchivoAleatorio
```

```
import java.io.Serializable;

public class RegistroCuentas implements Serializable

{

   private int cuenta;

   private String primerNombre;

   private String apellidoPaterno;

   private double saldo;



   // el constructor sin argumentos llama al otro constructor con valores
predeterminados

   public RegistroCuentas()

   {

      this( 0, "", "", 0.0 );

   }



   // inicializar un registro

   public RegistroCuentas( int cta, String nombre, String apellido,
double sald )

   {

      establecerCuenta( cta );

      establecerPrimerNombre( nombre );

      establecerApellidoPaterno( apellido );

      establecerSaldo( sald );

   }



   // establecer número de cuenta

   public void establecerCuenta( int cta )
```

```
{
   cuenta = cta;
}

// obtener número de cuenta
public int obtenerCuenta()
{
   return cuenta;
}

// establecer primer nombre
public void establecerPrimerNombre( String nombre )
{
   primerNombre = nombre;
}

// obtener primer nombre
public String obtenerPrimerNombre()
{
   return primerNombre;
}

// establecer apellido paterno
public void establecerApellidoPaterno( String apellido )
{
   apellidoPaterno = apellido;   }

// obtener apellido paterno
```

```
   public String obtenerApellidoPaterno()
   {
      return apellidoPaterno;
   }

   // establecer saldo
   public void establecerSaldo( double sald )
   {
      saldo = sald;
   }

   // obtener saldo
   public double obtenerSaldo()
   {
      return saldo;
   }

} // fin de la clase RegistroCuentas



// Subclase de RegistroCuentas para los programas que usan archivos de
acceso aleatorio.
import java.io.*;
public class RegistroCuentasAccesoAleatorio extends RegistroCuentas {

    public static final int TAMANIO = 72;

   // el constructor sin argumentos llama al otro constructor con los
valores predeterminados
```

```
   public RegistroCuentasAccesoAleatorio()

   {

      this( 0, "", "", 0.0 );

   }


   // inicializar un objeto RegistroCuentasAccesoAleatorio

   public RegistroCuentasAccesoAleatorio( int cuenta, String
primerNombre,

     String apellidoPaterno, double saldo )

   {

      super( cuenta, primerNombre, apellidoPaterno, saldo );

   }


   // leer un registro del objeto RandomAccecssFile especificado

   public void leer( RandomAccessFile archivo ) throws IOException

   {

      establecerCuenta( archivo.readInt() );

      establecerPrimerNombre( leerNombre( archivo ) );

      establecerApellidoPaterno( leerNombre( archivo ) );

      establecerSaldo( archivo.readDouble() );

   }

   // asegurarse que el nombre sea de la longitud apropiada

   private String leerNombre( RandomAccessFile archivo ) throws
IOException

   {

      char nombre[] = new char[ 15 ], temp;


      for ( int cuenta = 0; cuenta < nombre.length; cuenta++ ) {

         temp = archivo.readChar();
```

```
      nombre[ cuenta ] = temp;
   }

   return new String( nombre ).replace( '\0', ' ' );
}

// escribir un registro en el objeto RandomAccessFile especificado
public void escribir( RandomAccessFile archivo ) throws IOException
{
   archivo.writeInt( obtenerCuenta() );
   escribirNombre( archivo, obtenerPrimerNombre() );
   escribirNombre( archivo, obtenerApellidoPaterno() );
   archivo.writeDouble( obtenerSaldo() );
}

// escribir un nombre en el archivo; máximo 15 caracteres
private void escribirNombre( RandomAccessFile archivo, String nombre )
   throws IOException
{
   StringBuffer bufer = null;

   if ( nombre != null )
      bufer = new StringBuffer( nombre );
   else
      bufer = new StringBuffer( 15 );

   bufer.setLength( 15 );
   archivo.writeChars( bufer.toString() );
```

```
    }

} // fin de la clase RegistroCuentasAccesoAleatorio

//Ejemplo basico de como escribir y leer un archivo con java



public class FileHandler {

public static void main(String[ ] args) {
File file = new File("C:\\peper\\pepe.txt");

try {
writeFile(file);

} catch (IOException e) {
System.out.println("El archivo no pudo ser escrito o la ruta no existe");
}

try {
readFile(file);
} catch (IOException e) {
System.out.println("El archivo " + file.getPath() + "no pudo ser leido, " +
"compruebe que la ruta sea la correcta y que exista tal archivo");
}

}

private static void writeFile(File file) throws IOException{
FileWriter fw = new FileWriter(file);
BufferedWriter bw = new BufferedWriter (fw);
PrintWriter writer = new PrintWriter (bw);

writer.println("pepe,pepe2");
writer.println("alberto,alberto2");
writer.close();
}

private static void readFile(File file) throws IOException{
FileReader fr = new FileReader (file);
BufferedReader reader = new BufferedReader (fr);

//Se lee la primera línea
String linea = reader.readLine ();

while (linea != null)
{
System.out.println (línea);
//Se lee una nueva línea
línea = reader.readLine ();
}
reader.close();
}
}
```

TAREAS JAVA

1.-Crear una clase de alumno que incluya un método que capture 3 calificaciones. (consola o netbeans)

2.-Declarar una clase o registro de empleado, capturarlo y desplegarlo un número indeterminado de veces. (consola o netbeans)