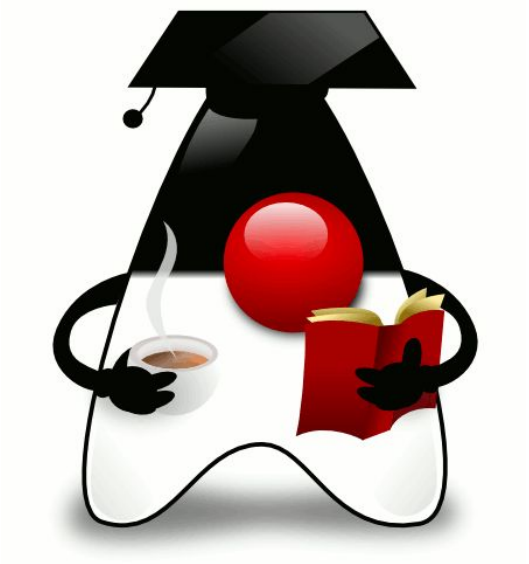
Clase 18: Manejo de archivos

Programación & Laboratorio III

Agenda

- **Concepto de archivo**
- Archivos en Java
- Concepto de buffering
- Escribir un archivo
- Leer un archivo
- Cerrar un archivo



Concepto de archivo

- Desde el punto de vista de más bajo nivel, podemos definir un archivo (o fichero) como un conjunto de bits almacenados en un dispositivo, y accesible a través de un camino de acceso (pathname) que lo identifica

Por qué usar archivos?

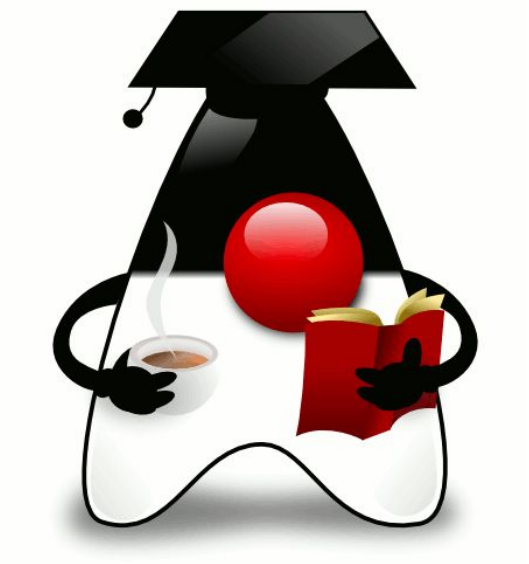
- Con frecuencia tendremos que guardar los datos de nuestro programa para poderlos recuperar más adelante. Los archivos se usan cuando el volumen de datos no es relativamente muy elevado.

Concepto de archivo (2)

- Los archivos suelen organizarse en estructuras jerárquicas de directorios. Estos directorios son contenedores de ficheros (y de otros directorios) que permiten organizar los datos del disco.
- El nombre de un archivo está relacionado con su posición en el árbol de directorios que lo contiene, lo que permite no sólo identificar unívocamente cada fichero, sino encontrarlo en el disco a partir de su nombre.
- Los archivos suelen tener una serie de metadatos asociados, como pueden ser su fecha de creación, la fecha de última modificación, su propietario o los permisos que tienen diferentes usuarios sobre ellos

Agenda

- ~~Concepto de archivo~~
- **Archivos en Java**
- Concepto de buffering
- Escribir un archivo
- Leer un archivo
- Cerrar un archivo



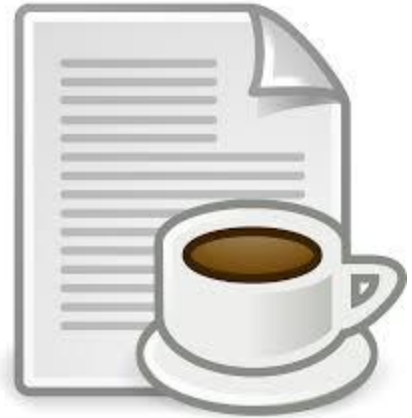
Archivos en Java

- La clase `File` en java representa un fichero y nos permite obtener información sobre él, como por ejemplo atributos, directorios, etc.
- Tiene tres constructores:
  - File(String path)
  - File(String path, String name)
  - File(File dir, String name)

El parámetro “path” indica el camino hacia el directorio donde se encuentra el archivo, y “name” indica el nombre del archivo.

Archivos en Java (2)

- Algunos métodos importantes:
  - String getName()
  - String getPath()
  - boolean exists()
  - boolean canWrite()
  - boolean canRead
  - boolean isFile()
  - boolean isDirectory()
  - long lastModified()
  - boolean renameTo(File dest);
  - boolean delete()
  - String[] list()



Archivos en Java (3)

- Al instanciar la clase `File` con un archivo, no se abre el fichero ni es necesario que este exista y precisamente esa es la primera información útil que podemos obtener del fichero: saber si existe o no.

```
import java.io.File;
...
File f = new File("path/mi_fichero.txt");
if (f.exists())
   // El fichero existe.
else
   // el fichero no existe.
```

Archivos en Java (4)

- Si el archivo existe, es decir, si la función exists() devuelve true, entonces podemos obtener información acerca del archivo. Por ejemplo:

```
    if(f.exists()){
       System.out.println("Nombre del archivo " + f.getName());
       System.out.println("Camino             " + f.getPath());
       System.out.println("Camino absoluto    " + f.getAbsolutePath());
       System.out.println("Se puede escribir  " + f.canRead());
       System.out.println("Se puede leer      " + f.canWrite());
       System.out.println("Tamaño             " + f.length());
    }
```

Archivos en Java (5)

- Si el archivo existe, podemos no tener permisos para leerlo, así que otra comprobación útil es si se puede hacer en él la operación deseada.

```
   if (f.canRead())
      // El archivo existe y se puede leer.

   if (f.canWrite())
      // El archivo existe y se puede escribir en él

   if (f.canExecute())
      // El archivo existe y se puede ejecutar
```

Archivos en Java (6)

- Tenemos, además, los correspondientes métodos `setReadable()`, `setReadOnly()`, `setWritable()` y `setExcutable()` que nos permiten cambiar las propiedades del archivo siempre que seamos los propietarios del mismo o tengamos permisos para hacerlo.

Archivos en Java (7)

- La siguiente comprobación útil que se puede hacer con un `File` es determinar si es un archivo normal o es un directorio. Si es un directorio, podemos obtener un listado de los ficheros contenidos en él.

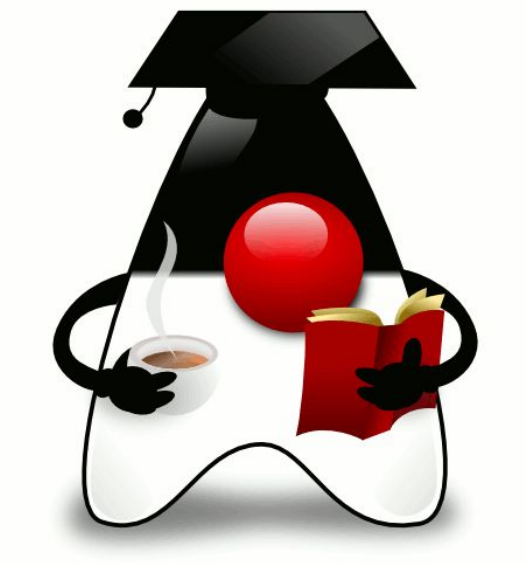
```
    // Se crea el File del directorio
    File directorio = new File("src/main/java/com/chuidiang/ejemplos/");

    // Si es un directorio
    if (directorio.isDirectory()) {
         // obtenemos su contenido
         File[] ficheros = directorio.listFiles();

         for (File fichero : ficheros) {
              System.out.println(fichero.getName());
         }
    }
```

Archivos en Java (8)

- Son archivos que podremos crear desde un programa Java y leer con cualquier editor de texto, o bien crear con un editor de textos y leer desde un programa Java.
- Para manipular archivos, siempre tendremos los mismos pasos:
  1) Abrir el archivo → Si no abrimos el archivo, obtendremos un mensaje de error al intentar acceder a su contenido.
  2) Escribir datos o leerlos.
  3) Cerrar el archivo → Si no cerramos el archivo, puede que realmente no se llegue a guardar ningún dato.

Agenda

- ~~Concepto de archivo~~
- ~~Archivos en Java~~
- **Concepto de buffering**
- Escribir un archivo
- Leer un archivo
- Cerrar un archivo



Concepto de Buffering

“Si no hubiera buffers, sería como comprar sin un carrito: debería llevar los productos uno a uno hasta la caja. Los buffers te dan un lugar en el que dejar temporalmente las cosas hasta que esté lleno. Por eso se realizarán menos viajes cuando usas el carrito”

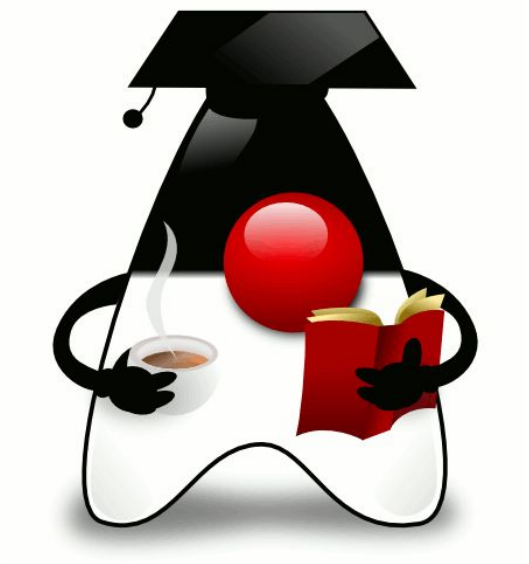
- Cualquier operación que implique acceder a memoria externa es muy costosa, por lo que es interesante reducir las operaciones de lectura/escritura que realizamos sobre los archivos.

Concepto de Buffering (2)

- Un buffer es una estructura de datos que permite el acceso por trozos a una colección de datos.
- Los buffers son útiles para evitar almacenar en memoria grandes cantidades de datos, en lugar de ello, se va pidiendo al buffer porciones pequeñas de los datos que se van procesando por separado. También resultan muy útiles para que la aplicación pueda ignorar los detalles concretos de eficiencia de hardware subyacente, la aplicación puede escribir al buffer cuando quiera, que ya se encargará el buffer de escribir a disco siguiendo los ritmos más adecuados y eficientes.

Agenda

- ~~Concepto de archivo~~
- ~~Archivos en Java~~
- ~~Concepto de buffering~~
- **Escribir un archivo**
- Leer un archivo
- Cerrar un archivo



Escribir archivo

- Para abrir un archivo usaremos un BufferedWriter, que se apoya en un FileWriter, que a su vez usa un "File" al que se le indica el nombre del archivo

```
BufferedWriter f = new BufferedWriter(
            new FileWriter(new File("mi_archivo.txt")));
```

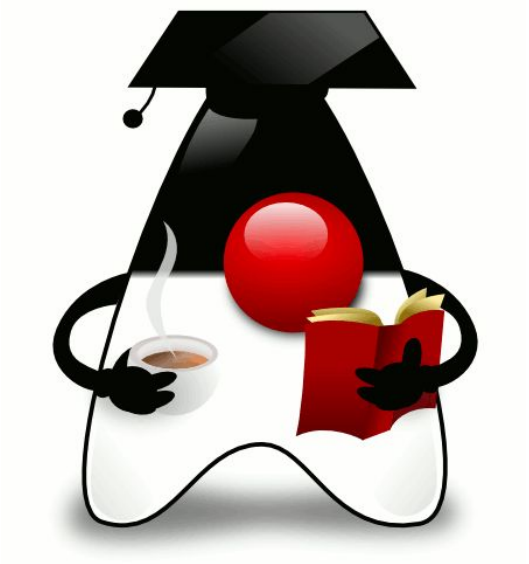
- En el caso de un archivo de texto, no escribiremos con "println()", como hacíamos en pantalla, sino con "write()". Cuando queramos avanzar a la línea siguiente, deberemos usar "newLine()":

Escribir archivo (2)

```
try {
    BufferedWriter fSalida = new BufferedWriter(new FileWriter(new File("mi_archivo.txt")));
    fSalida.write("Hola");
    fSalida.newLine();
    fSalida.write("Mundo!");
    fSalida.newLine();
    fSalida.close();
} catch (IOException e) {
    System.out.println("Se produjo un error al escribir en el archivo: " + e.getMessage());
}
```

Agenda

- ~~Concepto de archivo~~
- ~~Archivos en Java~~
- ~~Concepto de buffering~~
- ~~Escribir un archivo~~
- **Leer un archivo**
- Cerrar un archivo



Leer archivo

- Para leer de un fichero de texto usaremos "readLine()", que nos devuelve un String. Si ese String es null, quiere decir que se ha acabado el archivo y no se ha podido leer nada. Por eso, lo habitual es usar un "while" para leer todo el contenido de un fichero.
- Existe otra diferencia con la escritura: no usaremos un BufferedWriter, sino un BufferedReader, que se apoyará en un FileReader:

```
BufferedReader fEntrada = new BufferedReader(
                    new FileReader(new File("mi_archivo.txt")));
```

Leer archivo (2)

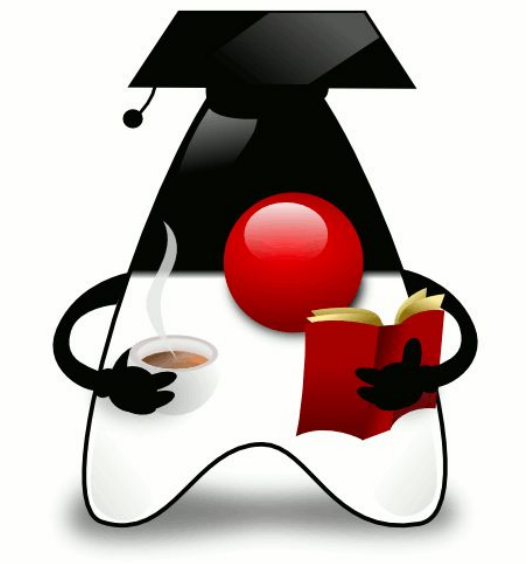
- La clase `java.io.BufferedReader` resulta ideal para leer archivos de texto y procesarlos. Permite leer eficientemente caracteres aislados, arrays o líneas completas como Strings. Cada lectura a un BufferedReader provoca una lectura en el archivo correspondiente al que está asociado. Es el propio BufferedReader el que se va encargando de recordar la última posición del fichero leído, de forma que posteriores lecturas van accediendo a posiciones consecutivas del fichero.

Leer archivo (3)

```
if (!(new File("mi_archivo.txt")).exists()) {
     return;
}
try {
     BufferedReader fEntrada = new BufferedReader(new FileReader(new File("mi_archivo.txt")));
     String linea=null;
     while ((linea=fEntrada.readLine()) != null) {
          System.out.println(linea);
     }
     fEntrada.close();
 } catch (IOException e) {
      System.out.println("Se produjo un error al escribir en el archivo: " + e.getMessage());
 }
```

Agenda

- ~~Concepto de archivo~~
- ~~Archivos en Java~~
- ~~Concepto de buffering~~
- ~~Escribir un archivo~~
- ~~Leer un archivo~~
- **Cerrar un archivo**



Cerrar

```
if (!(new File("mi_archivo.txt")).exists()) {
     return;
}
try {
     BufferedReader fEntrada = new BufferedReader(new FileReader(new File("mi_archivo.txt")));
     String linea=null;
     while ((linea=fEntrada.readLine()) != null) {
          System.out.println(linea);
     }
     fEntrada.close();
 } catch (IOException e) {
      System.out.println("Se produjo un error al escribir en el archivo: " + e.getMessage());
 }
```

Cerrar

```
if (!(new File("mi_archivo.txt")).exists()) {
     return;
}
try {
     BufferedReader fEntrada = new BufferedReader(new FileReader(new File("mi_archivo.txt")));
     String linea=null;
     while ((linea=fEntrada.readLine()) != null) {
          System.out.println(linea);
     }
 } catch (IOException e) {
      System.out.println("Se produjo un error al escribir en el archivo: " + e.getMessage() );
 } finally {
      fEntrada.close();
}
```

Bibliografía oficial

- https://docs.oracle.com/javase/tutorial/essential/io/file.html
- https://docs.oracle.com/javase/7/docs/api/java/io/File.html