

An abstract graphic on the left side of the slide. It features a close-up, angled view of a computer keyboard. Several keys are highlighted with glowing arrows in various colors (white, yellow, orange, green, blue). The background is a gradient from dark blue at the top to bright orange at the bottom, with a soft glow emanating from the bottom right corner.

GESTIÓN DE FICHEROS

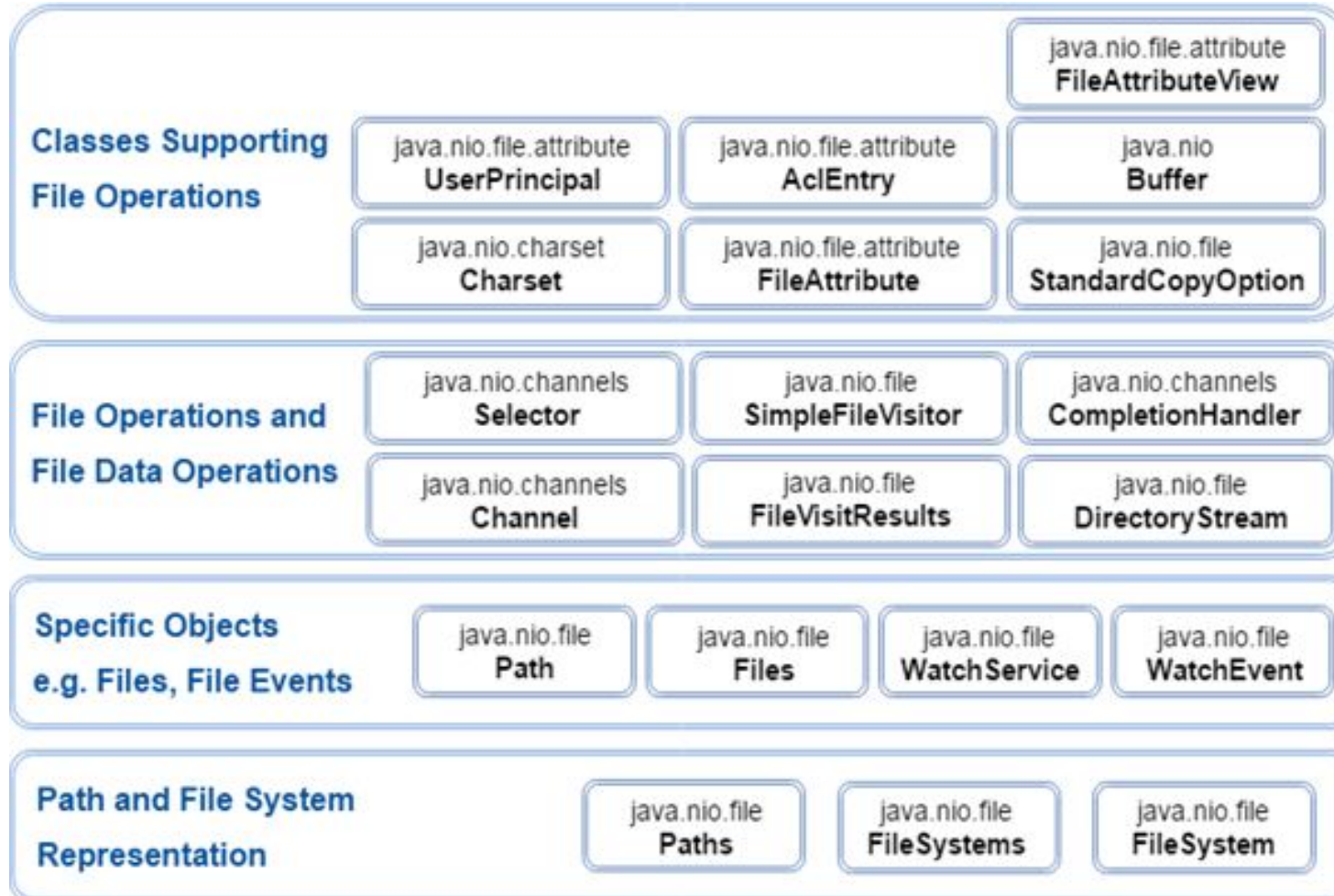
Módulo: Acceso a datos

Ciclo: Desarrollo de Aplicaciones
Multiplataforma

Introducción

- ♦ Hasta ahora hemos utilizado Java.IO.
 - ♦ Java.NIO disponible a partir de Java7:
 - ♦ Mejor rendimiento
 - ♦ Simplifica el manejo
 - ♦ Características:
 - ♦ **Canales y buffers:**
 - ♦ IO (secuencia bytes - caracteres).
 - ♦ NIO (lee canal □ escribe búffer)
 - ♦ E/S sin bloqueo □ un hilo pide al canal que escriba datos en el buffer mientras hace otra cosa.
 - ♦ **Selectores:** un hilo puede manejar múltiples canales
-

Java.NIO



Conceptos: Canal

- ◆ Encargado de conectarse a la fuente de datos.
- ◆ No es un flujo sino una puerta al almacén de datos.
- ◆ Su eficiencia depende del SO ya que cada JVM depende de cómo esté escrito el paquete NIO.
- ◆ **Instanciación:** no se usa new sino el patrón factoría.
- ◆ **Patrón factoría:** permite independizar interfaces y clases abstractas de clases finales.
 - ◆ Fábrica ⇒ Jerarquía Stream ⇒ reestructura de la biblioteca java.io
 - ◆ Canal ⇒ Clase FileChannel

```
FileInputStream istream = new FileInputStream(ruta);
```

```
FileChannel channel = istream.getChannel() ;
```

Conceptos: Buffer

- ◆ Encargado de:
 - ◆ Introducir y extraer información del canal.
 - ◆ Ofrecer utilidades para el intercambio de datos
- ◆ **Instanciación:** no se usa new sino el patrón factoría.
- ◆ **Patrón factoría:** permite independizar interfaces y clases abstractas de clases finales.
 - ◆ Clase principal \Rightarrow Buffer
 - ◆ Responsable del intercambio \Rightarrow ByteBuffer

Conceptos: Buffer

- ◆ Dos formas de instanciación:

- // Opción 1:

- ByteBuffer byteBuffer = ByteBuffer.allocate (1024) ;

- // Opción 2:

- ByteBuffer byteBuffer = ByteBuffer.allocateDirect (1024) ;

- ◆ **allocateDirect**: genera instancias más ligeras siendo más eficiente.

- ◆ **allocate**: gasta menos recursos de memoria y procesamiento

- ◆ Lectura y escritura: *getTipoDato* o *putTipoDato*

- ◆ Se pueden crear buffer de tipos específicos

- // Buffer original

- ByteBuffer bufferOriginal = ByteBuffer.allocate(MIDA_MAXIMA);

- // Buffer de tipo long

- LongBuffer bufferDeLongs = bufferOriginal.asLongBuffer();

Interfaz NIO Path

- ◆ Representa rutas
- ◆ Permite localizar fichero en el sistema de ficheros.
- ◆ Facilita el manejo de rutas Linux y Windows dependiendo del SO

```
Path p = Paths.get("/home/Escritorio/fichero.txt");
```

- ◆ No es necesario que exista el fichero.

FileSystem y Path

- ◆ Diferencias entre FileSystem y Path:
- ◆ **FileSystem**: define un sistema de ficheros completo
- ◆ **Path**: hace referencia a un directorio, fichero o enlace dentro de un sistema de ficheros.

Clase Java.NIO.Files

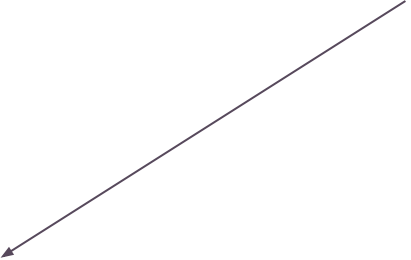
- ♦ Punta de entrada a la librería de ficheros java
 - ♦ Permite manejar ficheros reales del disco
 - ♦ Dispone de método estáticos que trabajan sobre objetos Path
 - ♦ Operaciones principales:
 - ♦ Verificar existencia y accesibilidad
 - ♦ Borrar un archivo o directorio
 - ♦ Copiar un archivo o directorio
 - ♦ Mover un archivo o directorio
-

Path

- ♦ .isAbsolute()
- ♦ .getFileName()
- ♦ .getParent()

FileSystem

- ♦ .getFileName()
- ♦ .getPath() □ usar .iterator() para obtener nombre ficheros



```
Path rutaDirectorio = sistemaFicheros.getPath("C:\\Users\\alumno");
Iterator<Path> it = rutaDirectorio.iterator();

while (it.hasNext()) {
    System.out.println(it.next().getFileName());
}
```

Files

- ♦ Files.exists(path)
- ♦ Files.isReadable(path)
- ♦ Files.isWritable(path)
- ♦ Files.isExecutable(path)
- ♦ Files.delete(path)
- ♦ Files.createDirectory(path)
- ♦ Files.copy(pathOrigen, pathDestino)
- ♦ Files.move(pathOrigen, pathDestino)

Escribir contenido en un fichero

- ♦ Desde Java podemos hacer más estrictos los permisos de accesos a los ficheros.
- ♦ Por ejemplo, si un usuario puede leer y escribir sobre un fichero podemos diseñar el programa Java para que solo lo abra en modo lectura.
- ♦ Usamos para ello el parámetro **OpenOptions** dentro del constructor `java.nio.Files`

OpenOptions

- ♦ La forma más fácil de utilizado es con el enum StandardOpenOptions.
- ♦ Tiene la siguiente estructura:

StandarOpenOption.*opcion*

- ♦ Donde opción puede ser:
- ♦ **WRITE**: habilita la escritura en el fichero.
- ♦ **APPEND**: todo lo que se escriba en el fichero se añadirá al final del mismo.
- ♦ **CREATE_NEW**: crea un fichero nuevo y lanza una excepción si ya existía.
- ♦ **CREATE**: crea el fichero si no existe y simplemente lo abre si ya existía.
- ♦ **TRUNCATE_EXISTING**: si el fichero existe, y tiene contenido, se ignora su contenido para sobrescribirlo desde el principio.

Lectura y escritura

♦ *Desde array de bytes*

```
Path inputFile = Paths.get("C:\\Users\\alumno\\FileEjemplo\\origen\\hola.txt");
```

```
Byte[] contenido = Files.readAllBytes(inputPath);
```

```
Files.write(outputPath, contenido, OpenOptions*) // StandardOpenOption.WRITE, StandardOpenOption.CREATE
```



♦ *Desde buffer*

```
BufferedReader inputReader = Files.newBufferedReader(inputPath, Charset.defaultCharset());
```

```
BufferedWriter outputWriter = Files.newBufferedWriter(outputPath, Charset.defaultCharset(),  
OpenOptions*);
```

```
String lineaString = inputReader.readLine();
```

```
outputWriter.write(lineaString, posición, longitud);
```

◆ Encargados de leer documentos XML:

◆ Analizadores secuenciales

- ◆ Extraen el contenido a medida que van descubriendo las etiquetas de apertura y cierre
- ◆ Ventaja: Son más rápidos
- ◆ Desventaja: es necesario recorrer el documento cada vez que se accede al contenido
- ◆ Ej.: SAX o Stax

◆ Analizadores jerárquicos

- ◆ Análisis secuencial que permite clasificar y almacenar en RAM el contenido
- ◆ Ventaja: facilitan la repetición de consultas y son muy eficientes.
- ◆ Ej.: DOM

♦ Recomendación del W3C

♦ Leer XML

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance ;  
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder;  
Document doc = dBuilder.parse( new File( "filter.xml" ));
```

♦ Escribir DOM a XML

```
// Creación de una instancia Transformer  
Transformer trans = TransformerFactory.newInstance.newTransformer;  
  
// Creación de los adaptadores Source y Results a partir de un Documento y un File.  
DOMSource source = new DOMSource (doc) ;  
StreamResult result = new StreamResult (file) ;  
trans.transform (source, result) ;
```

Estructura DOM

- **Document:** documento XML completo. Permite crear nuevos nodos.
 - **Node:** representa cualquier nodo del documento.
 - **Element:** expone propiedades y métodos para manipular los elementos del documento y sus atributos.
 - **Attr:** representa un atributo de un elemento.
 - **Text:** representa el contenido de un elemento o atributo.
 - **NodeList:** colección de nodos a los que se accede por medio de un índice.
-

Métodos DOM

- **Document.getDocumentElement():** retorna el elemento raíz del documento.
- **Node.getFirstChild():** retorna el primer nodo hijo del nodo.
- **Node.getLastChild():** retorna el ultimo nodo hijo del nodo.
- **Node.getNextSibling():** retorna el siguiente hermano de un nodo.
- **Node.getPreviousSibling():** retorna el hermano anterior de un nodo.
- **Node.getAttribute(attrName):** retorna el atributo del nodo cuyo nombre sea igual al valor pasado como parámetro.
- **Element.getElementsByTagName(String etiqueta):** busca dentro de un nodo

Métodos DOM

- **appendChild():** añadir nuevos hijos
- **setAttribute():** asignar el valor a un atributo
- **getAttribute():** consulta del valor de los atributos
- **getParentNode():** navegar por los nodos del árbol
- **getFirstChild/getLastChild():** obtener el padre
- **getNextSiblingpara():** obtener el primer / último hijo
- **getTextContent():** obtiene el contenido del elemento

Ejemplo 1: Lectura

```
File inputFile = new File("clase.xml");

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(inputFile);

doc.getDocumentElement().normalize();
System.out.println("Root element : " +
doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("alumno");

System.out.println("-----");

for (int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    System.out.println("\nCurrent Element : " + nNode.getNodeName());

    if (nNode.getNodeType() == Node.ELEMENT_NODE) {

        Element eElement = (Element) nNode;

        System.out.println("numero de alumno : " + eElement.getAttribute("numero"));
        System.out.println("nombre : " +
eElement.getElementsByTagName("nombre").item(0).getTextContent());
        System.out.println("apellido : " +
eElement.getElementsByTagName("apellido").item(0).getTextContent());
        System.out.println("apodo : " +
eElement.getElementsByTagName("apodo").item(0).getTextContent());
        System.out.println("marcas :
"+eElement.getElementsByTagName("marcas").item(0).getTextContent());
    }
}
```

Ejemplo 2: Creación

```
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
DOMImplementation implementation = builder.getDOMImplementation();
Document doc = docBuilder.createDocument(null, "root", null);

//Elemento raíz
Element rootElement = doc.getDocumentElement();

//Primer elemento
Element elemento1 = doc.createElement("elemento1");
rootElement.appendChild(elemento1);

//Se agrega un atributo al nodo elemento y su valor
Attr attr = doc.createAttribute("id");
attr.setValue("valor del atributo");
elemento1.setAttributeNode(attr);

Element elemento2 = doc.createElement("elemento2");
elemento2.setTextContent("Contenido del elemento 2");
rootElement.appendChild(elemento2);

//Se escribe el contenido del XML en un archivo
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(new File("/ruta/prueba.xml"));
transformer.transform(source, result);
```