

Ejemplos y actividades UD1. Manejo de ficheros

PROYECTO	CLASES-METODOS	OBSERVACIONES
GESTIÓN DE FICHEROS		
VerDir	Constructor => File(String directorioyfichero) Método => String[] list() Método => boolean isFile() Método => boolean isDirectory()	Muestra los ficheros (archivos o directorios) de un determinado directorio que se cargan en un array de elementos tipo String usando el método list()
Actividad1_1_Parte1	Constructor => File(String directorioyfichero) Método => File[] listFiles()	Muestra los ficheros (archivos o directorios) de un determinado directorio que se cargan en un array de elementos tipo File usando el método list()
Actividad1_1_Parte2	Constructor => File(String directorioyfichero) Método => String[] list()	El nombre del directorio a listar se pasa como parámetro al método main
VerInf	Constructor => File(String directorioyfichero) Método => String getName () Método => String getPath () y otros métodos que informan de atributos del fichero	Muestra varios atributos de un fichero (nombre, ruta, si se puede leer, etc.) Al especificar la ruta en Windows, al poner el carácter \ hay que añadir el carácter de escape \ por lo que tendré \\. Ejemplo E:\\Mis_documentos_Curso 2021-22\\fichero.txt
CrearDir	Constructor => File(String directorioyfichero) Constructor => File(File directorio, String fichero) Método => boolean mkdir() Método => boolean createNewFile() Método => renameTo(File nuevonombre)	Crea un directorio en el directorio actual (el del proyecto) y en ese directorio crea ficheros. Después renombra uno de los ficheros. Debido a que el método createNewFile puede lanzar IOException, se debe incluir en bloque try-catch.
RenombrarFichero	Constructor => File(String directorioyfichero) Método => renameTo(File nuevonombre)	Renombrar fichero
GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS DE TEXTO		
LeerFicTexto	Constructor declara fichero => File(String directorioyfichero) Constructor flujo de entrada => FileReader(File fichero) Método lee carácter a carácter como un entero => int read() Método cerrar fichero => int close()	Lee uno a uno los caracteres de un fichero de texto y los muestra en pantalla. El método read() devuelve -1 cuando no hay carácter. Debido a que el método read() puede lanzar IOException, se debe incluir en bloque try-catch, pero en este caso, como alternativa al try-catch, en este ejemplo añade throws IOException al método main()
Actividad1_2	Constructor declara fichero => File(String directorioyfichero) Constructor flujo de entrada => FileReader(File fichero) Método lee carácter a carácter como un entero => int read() Método cerrar fichero => int close()	Muestra en la consola el contenido de un fichero de texto que se le pasa al programa como un parámetro.
EscribirFicTexto	Constructor declara fichero => File(String directorioyfichero) Constructor flujo de salida borrando => FileWriter(File fichero) Constructor flujo de salida para adjuntar => FileWriter(File fichero, boolean append) Método escribe carácter (pe. "\n") => write(char c) Método escribe array de caracteres => write(char []) Método adjuntar carácter => append(char c) Método cerrar fichero => int close()	Escribe caracteres en un fichero de varias formas. Si no existe el fichero, al crear el flujo de salida se crea el fichero. 1. los caracteres se escriben uno a uno a partir de un String que se convierte en array de caracteres. 2. se escribe un array de caracteres 3. se escribe un String 4. escribe un array de Strings
LeerFichTextoBuf	Constructor flujo de entrada ==> BufferedReader(FileReader(File fichero)) Método lee línea => String readLine() Método cerrar fichero => int close()	Lee líneas completas de texto de un fichero determinado ubicado en el mismo directorio que el proyecto y las muestra en pantalla. El método readLine() devuelve null cuando no hay línea La lectura se incluye en bloque try-catch.
EscribirFichTextoBuf	Constructor flujo de salida ==> BufferedWriter(FileReader(File fichero)) Método escribe línea => Write(String str) Método escribe salto de línea ==> newLine() Método cerrar fichero => int close()	Escribe líneas completas de texto y añade el salto de línea
GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS BINARIOS (ocupan menos espacio en disco, pero no son legibles por editores de texto)		
EscribirFichBytes	Constructor flujo salida => FileOutputStream(File file) => FileOutputStream(File file, boolean append) Constructor flujo entrada => FileInputStream(File file) Método escribe un byte => write(int b) Método lee un byte => read(int b) Método cerrar stream => int close()	Escribe bytes en un fichero (si no existe lo crea el constructor) y después los visualiza. Al elegir el constructor del flujo de salida se puede optar por adjuntar datos al fichero ya existente (append = true) El método read() devuelve -1 cuando no hay dato a leer.

LeerFichBytes	<p>Constructor flujo entrada => <code>FileInputStream(File file)</code></p> <p>Método lee un byte => <code>read(int b)</code></p> <p>Método cerrar stream => <code>int close()</code></p>	<p>Lee los bytes de un fichero y los visualiza.</p> <p>El método <code>read()</code> devuelve -1 cuando no hay dato a leer.</p>
EscribirFichData	<p>Constuctor flujo salida =></p> <p><code>DataOutputStream(OutputStream out)</code></p> <p>Método escribe dato (según tipo de dato) => <code>writeXXX(tipoDato dato)</code></p> <p>Método cerrar stream => <code>int close()</code></p>	<p>Escribe en un fichero de datos tipos primitivos (int, string) que están en un array de elementos del tipo correspondiente.</p>
LeerFichData	<p>Constructor flujo entrada => <code>DataInputStream(InputStream in)</code></p> <p>Método lee dato (según tipo de dato) => <code>readXXX(tipoDato dato)</code></p> <p>Método cerrar stream => <code>int close()</code></p>	<p>Lee datos tipos primitivos (int, string) desde un fichero de datos. La lectura de datos debe hacerse sabiendo el tipo y orden de los datos que están escritos en el fichero. Para controlar el fin de la lectura de datos utiliza un bucle while dentro de un try-catch(EOFException)</p>
GESTIÓN DE FLUJOS DE OBJETOS DESDE/HACIA FICHEROS BINARIOS (guardando todos los atributos del objeto. El objeto debe implementar la interfaz Serializable convierte el objeto en una secuencia de bits.) Interfaz Serializable		
<p>EscribirFichObject</p> <p>Persona</p> <p>EscribirFichObject</p> <p>(permite adjuntar nuevos objetos al fichero)</p>	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor flujo salida => <code>FileOutputStream(File file)</code></p> <p><code>=> FileOutputStream(File file, boolean append)</code></p> <p>Constructor enlaza flujo datos (stream) al flujo de bits => <code>ObjectOutputStream(OutputStream out)</code></p> <p>Método escribe objeto => <code>writeObject(Object obj)</code></p> <p>Método cerrar stream => <code>int close()</code></p>	<p>Define el objeto Persona que implementa la interfaz Serializable y el programa escribe en un fichero de datos los atributos correspondientes al objeto Persona. Cuando adjuntamos objetos a un fichero existente aparece un problema porque se añade una cabecera al fichero java.io.StreamCorruptedException: invalid type code: AC</p> <p>Para que no se escriba la cabecera es necesario sobrescribir (extends) la clase ObjectOutputStream que en mi caso lo he hecho creando la clase MiObjectOutputStream</p> <p>En el programa, Si el fichero no existe se crea un objeto ObjectOutputStream que crea cabecera, pero si el fichero ya existe, se llama a la clase miObjectOutputStream que no escribe cabecera.</p>
<p>LeerFichObject</p> <p>Persona</p>	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor flujo entrada =></p> <p><code>ObjectInputStream(InputStream in)</code></p> <p>Método lee objeto => <code>readObject()</code></p> <p>Método cerrar stream => <code>int close()</code></p>	<p>Lee los objetos desde un archivo de datos. Al leer los datos con el método <code>readObject</code> hace un cast a la clase del Objeto, pe. (Persona)</p>
GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS BINARIOS DE ACCESO ALEATORIO		
EscribirFichAleatorio	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor crea fichero acceso aleatorio =></p> <p><code>RandomAccessFile(File file, String mode) modo "rw"</code></p> <p>Métodos de escritura de datos según su tipo =></p> <p><code>writeInt(int v)</code></p> <p><code>writeChars(String s)</code></p> <p><code>writeDouble(double v)</code></p>	<p>Abre el fichero en modo "rw" e inserta registros con datos de empleado (total 36 bytes):</p> <ul style="list-style-type: none"> - Identificador (entero = 4 bytes) - Apellido (10 caracteres UNICODE, 2 bytes x carácter) - Nº Departamento (entero = 4 bytes) - Salario (double = 8 bytes) <p>Los datos se escriben desde un array con elementos de cada uno de los tipos y el identificador es el índice del array + 1.</p> <p>En el caso del apellido para que todos ocupen 10 caracteres utiliza un buffer.</p>
LeerFichAleatorio	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor crea fichero acceso aleatorio =></p> <p><code>RandomAccessFile(File file, String mode) modo "r"</code></p> <p>Método para posicionarse en un registro => <code>seek(long pos)</code></p> <p>Método para saber la posición del puntero => <code>getFilePointer()</code></p> <p>Método pasa saber el tamaño del fichero => <code>length()</code></p> <p>Método desplaza puntero unos bytes => <code>skipBytes(int n)</code></p> <p>Métodos de lectura de datos según su tipo =></p> <p><code>readInt()</code>, <code>readDouble()</code>, <code>readChar()</code></p>	<p>Realiza la lectura de los registros de un fichero aleatorio. Para la lectura de los datos debemos controlar la posición del puntero dentro del fichero (método <code>getFilePointer()</code>), sabiendo el tamaño de cada registro y controlando el tamaño del fichero (método <code>length()</code>). El desplazamiento dentro del fichero se realiza con el método <code>seek(posición)</code>.</p>
EscribirFichAleatorioUnReg	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor crea fichero acceso aleatorio =></p> <p><code>RandomAccessFile(File file, String mode) modo "rw"</code></p> <p>Método para posicionarse en un registro => <code>seek(long pos)</code></p> <p>Métodos de escritura de datos según su tipo =></p> <p><code>writeInt(int v)</code>, <code>writeChars(String s)</code>, <code>writeDouble(double v)</code></p>	<p>Inserta un registro en un fichero aleatorio del que conoce su estructura (tamaño del registro). Calcula la posición y se posiciona con el método <code>seek(posición)</code></p>
LeerFichAleatorioUnReg	<p>Constructor declara fichero => <code>File(String directorioyfichero)</code></p> <p>Constructor crea fichero acceso aleatorio =></p> <p><code>RandomAccessFile(File file, String mode) modo "rw"</code></p> <p>Método para posicionarse en un registro => <code>seek(long pos)</code></p>	<p>Lee un registro de un fichero aleatorio. El número del registro se pasa como parámetro al programa.</p>

	Métodos de escritura de datos según su tipo => <code>writeInt(int v)</code> , <code>writeChars(String s)</code> , <code>writeDouble(double v)</code>	
Actividad1_3_Consulta		
Actividad1_3_Insercion		
Actividad1_4_Modificacion		
Actividad1_4_Borrado		
TRABAJO CON FICHEROS XML => https://www.w3schools.com/xml/xml_what.asp Procesadores XML (Parser) => https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm Documentación Oracle Java SE8 => https://docs.oracle.com/javase/8/docs/ Documentación (API DOM) paquete org.w3c.dom => https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/package-tree.html Documentación (API SAX) paquete org.xml.sax => http://www.saxproject.org/apidoc/org/xml/sax/package-tree.html Documentación (API XStream) XStream Core 1.4.18 => http://x-stream.github.io/javadoc/index.html		
CrearEmpleadoXml	Construye el procesador o parser DOM => <code>DocumentBuilderFactory()</code> (try -Catch controla ParserConfigurationException) Crea una clase para crear el documento XML => <code>DocumentBuilder()</code> Crea un interfaz para usar el documento DOM => <code>DOMImplementation</code> Para trabajar con el documento XML (en memoria) emplea los métodos del interfaz <code>DOMImplementation</code> Convierte el documento en un fichero => https://docs.oracle.com/javase/tutorial/jaxp/xslt/writingDom.html Para realizar la conversión Crea una clase Transformer a partir de <code>TransformerFactory</code>	Crea un fichero XML a partir de los datos de un fichero aleatorio existente, utilizando el parser DOM Recorre los registros del fichero aleatorio donde están los datos y crea en memoria la estructura de árbol del documento XML (raíz, nodos, ...) Finalmente convierte el documento XML en un fichero. ----- Comentario: Los constructores <code>DocumentBuilder</code> y <code>TransformerFactory</code> están protegidos. Para usarlos hay que crear antes una instancia de la clase => https://stackoverflow.com/questions/1057221/what-are-practical-uses-of-a-protected-constructor/1057245
LecturaEmpleadoXml	Construye el procesador o parser DOM => <code>DocumentBuilderFactory()</code> (try -Catch controla ParserConfigurationException) Método para leer el documento XML analizándolo y convirtiéndolo en documento => <code>parse(InputStream is)</code> Para trabajar con el documento XML (en memoria) emplea los métodos del interfaz <code>Document</code>	Realiza la lectura de un fichero XML, utilizando el parser DOM. El fichero se procesa con el parser DOM y una vez que está en memoria se emplean los métodos para leer etiquetas, valores y demás elementos del archivo.
Actividad1_5 FichPersona.dat		Lee los datos de un fichero .dat que contiene objetos Persona y a partir de él crea un fichero XML utilizando el parser DOM.
UD1_PruebaSax1 alumnos.xml	Construye el procesador o parser SAX => <code>XMLReaderFactory()</code> (XMLReaderFactory está deprecated y se recomienda usar <code>SAXParserFactory</code>) Crea procesador para el fichero => <code>createXMLReader()</code> Crea un gestor de contenido personalizado. Personaliza el procesador SAX al sobrescribir los métodos de <code>DefaultHandler()</code> para tratar los eventos que produce la lectura del fichero (XML) Método asigna gestor al procesador => <code>setContentHandler(ContentHandler handler)</code> Declara el archivo a procesar => <code>InputSource()</code> Método ejecuta el procesador => <code>parse(InputSource input)</code>	Realiza la lectura de un fichero XML, utilizando un parser SAX, en concreto XMLReaderFactory, pero que está deprecated. Los métodos para tratar el archivo se definen sobrescribiendo la clase <code>DefaultHandler</code> https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html
UD1_PruebaSax2 alumnos.xml	Idem a UD1_PruebaSax1, pero usando SAXParserFactory en lugar de XMLReaderFactory que está deprecated.	Realiza la lectura de un fichero XML, utilizando un parser SAX, en concreto SAXParserFactory
Actividad1_6 Empleados.xml		Realiza la lectura de un fichero XML, utilizando un parser SAX, en concreto SAXParserFactory
EscribirPersonas FichPersona.dat	Añadir archivos JAR al Build path: <code>xstream-1.4.18.jar</code> , <code>kxml2-min-2.3.0.jar</code> Crea el fichero XML, asignando valores a las etiquetas y los datos, empleando la clase <code>XStream</code> Método asigna nombre a la clase del objeto => <code>alias</code> Método asigna nombre a los nombres de un campo => <code>aliasField</code> Método para omitir el nodo raíz => <code>addImplicitCollection</code> Método que genera el fichero XML a partir de la lista de objetos => <code>toXML</code>	Crea una lista de objetos (ListaPersonas) del tipo (clase) Persona, que obtiene leyendo desde un fichero .dat que incluye los datos. Después convierte la lista de objetos (ListaPersonas) en un fichero XML, es decir, serializa objetos Java a XML. Lo hace empleando la librería XStream => https://x-stream.github.io/download.html
EscribirPersonas2 FichPersona.dat	Idem a EscribirPersonas, pero capturando excepciones con bloques try-catch	

LeerPersonas Personas.xml	<p>Añadir archivos JAR al Build path: xstream-1.4.18.jar, xpp3_min-1.1.4c.jar y xmlpull-1.1.3.1.jar</p> <p>Crea el fichero XML, asignando valores a las etiquetas y los datos, empleando la clase XStream</p> <p>Método asigna nombre a la clase del objeto => alias</p> <p>Método asigna nombre a los nombres de un campo => aliasField</p> <p>Método para omitir el nodo raíz => addImplicitCollection</p> <p>Método que obtiene la lista de objetos a partir del fichero XML => fromXML</p>	<p>Realiza la lectura de un fichero XML</p> <p>Deserializa objetos Java a partir de un fichero XML, empleando la librería XStream => https://x-stream.github.io/download.html</p> <p>Al código del ejemplo del libro, después de crear el objeto XStream con la siguiente línea: XStream xstream = new XStream(); hay que añadirle las dos líneas de código siguientes: xstream.allowTypes(new Class[] {ListaPersonas.class}); xstream.allowTypes(new Class[] {Persona.class}); para que se puedan crear las clases.</p>
Convertidor	<p>Para realizar la conversión Crea una clase Transformer a partir de TransformerFactory</p> <p>Para indicar la hoja de estilos usa el método => newTransformer (Source source)</p> <p>Para realizar la transformación usa el método => transform (Source ficheroXML, Source ficheroHTML)</p>	<p>Genera un fichero HTML a partir de un fichero XML y otro XSL (https://www.w3.org/Style/XSL/)</p> <p>Ficheros XSL (expresa hojas de estilo en lenguaje XML)</p>
DETECCIÓN Y TRATAMIENTO DE EXCEPCIONES => Ejemplos-tipos de excepciones		
UD1_EjemploExcepcion	Clase de la que se crea un objeto al producirse una excepción => Exception	Ejemplo de programa que produce una excepción (al dividir por 0 - by zero) y detiene el programa, indicando el error en la consola.
UD1_EjemploExcepciones	<p>Para capturar la excepción desde el programa se emplea el bloque Try- Catch.</p> <p>Es posible anidar bloques Try-Catch</p> <p>La clase Exception si se usa debe ponerse al final de todos los manejadores de eventos</p>	Ejemplo de programa que captura varios tipos de excepción.
UD1_EjemploExcepciones2	Los métodos de la clase Throwable permiten mostrar información de la excepción producida => Throwable	Ejemplo que utiliza varios métodos de la clase Throwable para mostrar información sobre la excepción que se produzca
TRABAJO CON FICHEROS XML con JAXB API JAXB => https://javaee.github.io/jaxb-v2/		
UD1_LecturaJAXB	<p>Se trabaja con el paquete java.xml.bind</p> <p>Añadir archivos JAR al Build path: istack-commons-runtime.jar, javax.activation-api.jar, jaxb-api.jar y jaxb-runtime.jar</p> <p>Clase que gobierna la deserialización => Unmarshaller</p>	Realiza la lectura de un documento XML y lo convierte en objetos (deserializa) de una clase definida previamente por el programador, empleando JAXB
UD1_EscrituraJAXB	<p>Se trabaja con el paquete java.xml.bind</p> <p>Añadir archivos JAR al Build path: istack-commons-runtime.jar, javax.activation-api.jar, jaxb-api.jar y jaxb-runtime.jar</p> <p>Clase que gobierna la serialización => Marshaller</p>	Realiza la escritura de un documento XML a partir de los valores de las instancias de una clase de objetos definidos previamente por el programador
Actividad1_7		Lectura de un fichero de datos que contiene objetos
Actividad1_8 departamentos.dat		Creación de fichero XML a partir de fichero de Objetos usando DOM
Actividad1_9 departamentos.dat		Creación de fichero XML a partir de fichero de Objetos usando XStream
Actividad1_10 departamentos.xml		Conversión de fichero XML a HTML
Actividad1_11 cine.xml		Lectura de fichero XML empleando JAXB
Actividad1_12 cine.xml		Escritura de fichero XML empleando JAXB