Ejemplos y actividades UD1. Manejo de ficheros

| PROYECTO | CLASES-METODOS | OBSERVACIONES |
|---|---|---|
| GESTIÓN DE FICHEROS | | |
| VerDir<br>UD1_Actividad1_1_Parte1 | Constructor => File(String directorioyfichero)<br>Método => String[] list()<br>Método => boolean isFile()<br>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<br>UD1_Actividad1_1_Parte1 | Constructor => File(String directorioyfichero)<br>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)<br>Método => String[] list() | El nombre del directorio a listar se pasa como parámetro al método main |
| VerInf | Constructor => File(String directorioyfichero)<br>Método => String getName ()<br>Método => String getPath ()<br>y otros métodos que informan de atributos del fichero | Muestra varios atributos de un fichero (nombre, ruta, si se puede leer, etc.)<br>Al especificar la ruta en Windows, al poner el carácter \ hay que añadir el carácter de escape \ por lo que tendré \\.<br>Ejemplo `E:\\Mis_documentos\\_Curso 2021-22\\fichero.txt` |
| CrearDir | Constructor => File(String directorioyfichero)<br>Constructor => File(File directorio, String fichero)<br>Método => boolean mkdir()<br>Método => boolean createNewFile()<br>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)<br>Método => renameTo(File nuevonombre) | Renombrar fichero |
| GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS DE TEXTO | | |
| LeerFicTexto | Constructor declara fichero => File(String directorioyfichero)<br>Constructor flujo de entrada => FileReader(File fichero)<br>Método lee carácter a carácter como un entero => int read()<br>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)<br>Constructor flujo de entrada => FileReader(File fichero)<br>Método lee carácter a carácter como un entero => int read()<br>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)<br>Constructor flujo de salida borrando => FileWriter(File fichero)<br>Constructor flujo de salida para adjuntar =><br>`FileWriter(File fichero, boolean append)`<br>Método escribe carácter (pe. "\n") => write(char c)<br>Método escribe array de caracteres => write(char [])<br>Método adjuntar carácter => append(char c)<br>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.<br>1. los caracteres se escriben uno a uno a partir de un String que se convierte en array de caracteres.<br>2. se escribe un array de caracteres<br>3. se escribe un String<br>4. escribe un array de Strings |
| LeerFichTextoBuf | Constructor flujo de entrada ==> BufferedReader(FileReader(File fichero)<br>Método lee línea => String readLine()<br>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 incluiye en bloque try-catch. |
| EscribirFichTextoBuf | Constructor flujo de salida ==> BufferedWriter(FileReader(File fichero)<br>Método escribe línea => Write(String str)<br>Método escribe salto de línea ==> newLine()<br>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)`<br>`=> FileOutputStream(File file, boolean append)`<br>Constructor flujo entrada => `FileInputStream(File file)`<br>Método escribe un byte => write(int b)<br>Método lee un byte => read(int b)<br>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)<br>El método read() devuelve -1 cuando no hay dato a leer. |

| | | |
|---|---|---|
| LeerFichBytes | Constructor flujo entrada => `FileInputStream(File file)`<br>Método lee un byte => read(int b)<br>Método cerrar stream => int close() | Lee los bytes de un fichero y los visualiza.<br>El método read() devuelve -1 cuando no hay dato a leer. |
| EscribirFichData | Constuctor flujo salida =><br>**DataOutputStream(OutputStream** out)<br>Método escribe dato (según tipo de dato) => writeXXXX(tipoDato dato )<br>Método cerrar stream => int close() | Escribe en un fichero de datos tipos primitivos (int, string) que están en un array de elementos del tipo correspondiente. |
| LeerFichData | Constructor flujo entrada => `DataInputStream(InputStream in)`<br>Método lee dato (según tipo de dato) => readXXXX(tipoDato dato )<br>Método cerrar stream => int close() | 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 ) |
| 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 | | |
| EscribirFichObject Persona<br><br>EscribirFichObject (permite adjuntar nuevos objetos al fichero) | Constructor declara fichero => File(String directorioyfichero)<br>Constructor flujo salida => `FileOutputStream(File file)`<br>`=> FileOutputStream(File file, boolean append)`<br>Constructor enlaza flujo datos (stream) al flujo de bits =><br>`ObjectOutputStream(OutputStream out)`<br>Método escribe objeto => `writeObject(Object obj)`<br>Método cerrar stream => int close() | Define el objeto Persona que implementa la interfaz Serializable y el programa escribe en un fichero de datos los atributos correspondientes al objeto Persona.<br>Cuando adjuntamos objetos a un fichero existente aparece un problema porque se añade una cabecera al fichero `java.io.StreamCorruptedException: invalid type code: AC`<br>Para que no se escriba la cabecera es necesario sobreescribir (extends) la clase ObjectOutputStream que en mi caso lo he hecho creando la clase MiObjectOutputStream<br>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. |
| LeerFichObject Persona | Constructor declara fichero => File(String directorioyfichero)<br>Constructor flujo entrada =><br>`ObjectInputStream(InputStream in)`<br>Método lee objeto => `readObject()`<br><br>Método cerrar stream => int close() | Lee los objetos desde un archivo de datos. Al leer los datos con el método readObject hace un cast a la clase del Objeto, pe. (Persona) |
| GESTIÓN DE FLUJOS DE DATOS DESDE/HACIA FICHEROS BINARIOS DE ACCESO ALEATORIO | | |
| EscribirFichAleatorio | Constructor declara fichero => File(String directorioyfichero)<br>Constructor crea fichero acceso aleatorio =><br>`RandomAccessFile(File file, String mode) modo "rw"`<br><br>Métodos de escritura de datos según su tipo =><br>`writeInt(int v)`<br>`writeChars(String s)`<br>`writeDouble(double v)` | Abre el fichero en modo "rw" e inserta registros con datos de empleado (total 36 bytes):<br>- Identificador (entero = 4 bytes)<br>- Apellido (10 caracteres UNICODE, 2 bytes x carácter)<br>- Nº Departamento (entero = 4 bytes)<br>- Salario (double = 8 bytes)<br>Los datos se escriben desde un array con elementos de cada uno de los tipos y el identificador es el indice del array + 1.<br>En el caso del apellido para que todos ocupen 10 caracteres utiliza un buffer. |
| LeerFichAleatorio | Constructor declara fichero => File(String directorioyfichero)<br>Constructor crea fichero acceso aleatorio =><br>`RandomAccessFile(File file, String mode) modo "r"`<br><br>Método para posicionarse en un registro => `seek(long pos)`<br>Método para saber la posición del puntero => `getFilePointer()`<br>Método pasa saber el tamaño del fichero => `length()`<br>Método desplaza puntero unos bytes => `skipBytes(int n)`<br>Métodos de lectura de datos según su tipo =><br>`readInt(),readDouble(),readChar()` | 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 getFilePointer), sabiendo el tamaño de cada registro y controlando el tamaño del fichero (método length). El desplazamiento dentro del fichero se realiza con el método seek(posición). |
| EscribirFichAleatorioUnReg | Constructor declara fichero => File(String directorioyfichero)<br>Constructor crea fichero acceso aleatorio =><br>`RandomAccessFile(File file, String mode) modo "rw"`<br>Método para posicionarse en un registro => `seek(long pos)`<br><br>Métodos de escritura de datos según su tipo =><br>`writeInt(int v),writeChars(String s), writeDouble(double v)` | Inserta un registro en un fichero aleatorio del que conoce su estructura (tamaño del registro). Calcula la posición y se posiciona con el método seek(posición) |
| LeerFichAleatorioUnReg | Constructor declara fichero => File(String directorioyfichero)<br>Constructor crea fichero acceso aleatorio =><br>`RandomAccessFile(File file, String mode) modo "rw"`<br>Método para posicionarse en un registro => `seek(long pos)` | Lee un registro de un fichero aleatorio. El número del registro se pasa como parámetro al programa. |

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

| | | |
|---|---|---|
| LeerPersonas<br>Personas.xml | Añadir archivos JAR al Build path: xtream-1.4.18.jar, xpp3_min-1.1.4c.jar y xmlpull-1.1.3.1.jar<br>Crea el fichero XML, asignando valores a las etiquetas y los datos, empleando la clase XStream<br>Método asigna nombre a la clase del objeto => alias<br>Método asigna nombre a los nombres de un campo => aliasField<br>Método para omitir el nodo raíz => addImplicitCollection<br>Método que obtiene la lista de objetos a partir del fichero XML => fromXML | Realiza la lectura de un fichero XML<br>Deserializa objetos Java a partir de un fichero XML, empleando la librería XStream => https://x-stream.github.io/download.html<br><br>Al código del ejemplo del libro, después de crear el objeto XStream con la siguiente línea:<br>XStream xstream = **new** XStream();<br>hay que añadirle las dos líneas de código siguientes:<br>xstream.allowTypes(**new** Class[] {ListaPersonas.**class**});<br>xstream.allowTypes(**new** Class[] {Persona.**class**});<br>para que se puedan crear las clases. |
| Convertidor | Para realizar la conversión Crea una clase Transformer a partir de `TransformerFactory`<br>Para indicar la hoja de estilos usa el método =><br>newTransformer (Source source)<br>Para realizar la transformación usa el método =><br>transform (Source ficheroXML, Source ficheroHTML) | Genera un fichero HTML a partir de un fichero XML y otro XSL (https://www.w3.org/Style/XSL/ )<br>Ficheros XSL (expresa hojas de estilo en lenguaje XML) |
| 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 | Para capturar la excepción desde el programa se emplea el bloque Try- Catch.<br>Es posible anidar bloques Try-Catch<br>La clase Exception si se usa debe ponerse al final de todos los manejadores de eventos | 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 | Se trabaja con el paquete java.xml.bind<br>Añadir archivos JAR al Build path: istack-commons-runtime.jar, javax.activation-api.jar, jaxb-api.jar y jaxb-runtime.jar<br>Clase que gobierna la deserialización => Unmarshaller | 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 | Se trabaja con el paquete java.xml.bind<br>Añadir archivos JAR al Build path: istack-commons-runtime.jar, javax.activation-api.jar, jaxb-api.jar y jaxb-runtime.jar<br>Clase que gobierna la serialización => Marshaller | 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<br>departamentos.dat | | Creación de fichero XML a partir de fichero de Objetos usando DOM |
| Actividad1_9<br>departamentos.dat | | Creación de fichero XML a partir de fichero de Objetos usando XStream |
| Actividad1_10<br>departamentos.xml | | Conversión de fichero XML a HTML |
| Actividad1_11<br>cine.xml | | Lectura de fichero XML empleando JAXB |
| Actividad1_12<br>cine.xml | | Escritura de fichero XML empleando JAXB |