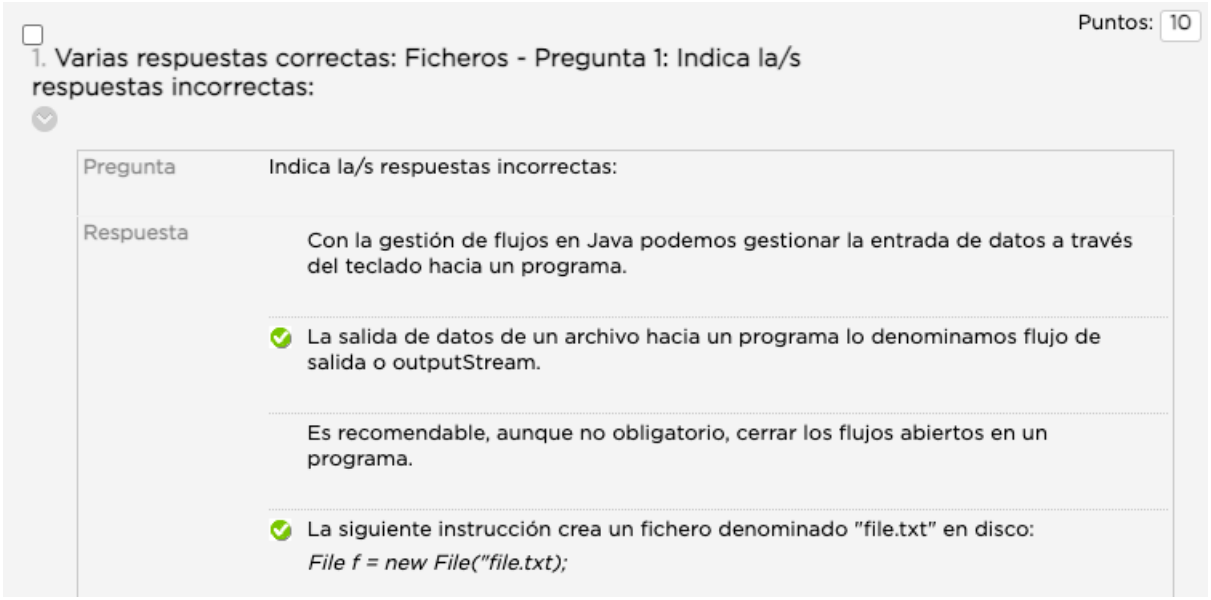
**Pregunta 1.**



Puntos: 10

1. Varias respuestas correctas: Ficheros - Pregunta 1: Indica la/s respuestas incorrectas:

| | |
|---|---|
| Pregunta | Indica la/s respuestas incorrectas: |
| Respuesta | Con la gestión de flujos en Java podemos gestionar la entrada de datos a través del teclado hacia un programa. |
| | La salida de datos de un archivo hacia un programa lo denominamos flujo de salida o outputStream. |
| | Es recomendable, aunque no obligatorio, cerrar los flujos abiertos en un programa. |
| | La siguiente instrucción crea un fichero denominado "file.txt" en disco: *File f = new File("file.txt);* |

Con la gestión de flujos en Java podemos gestionar la entrada de datos a través del teclado hacia un programa.

CORRECTO: La gestión de flujos es la forma que tiene Java de gestionar las entradas y salidas en los dispositivos. Por ejemplo, para un teclado.

La salida de datos de un archivo hacia un programa lo denominamos flujo de salida o outputStream.

INCORRECTO: Es justo al revés. Cuando salen datos de un programa a un dispositivo es cuando lo denominamos flujo de salida.

Es recomendable, aunque no obligatorio, cerrar los flujos abiertos en un programa.

CORRECTO: Es recomendable. Muy recomendable, pero no obligatorio. Nuestro programa funcionaría igual, pero es menos eficiente.

La siguiente instrucción crea un fichero denominado "file.txt" en disco:

File f = new File("file.txt);

INCORRECTO: Para crear un archivo en disco (físicamente) hay que llamar al método *createNewFile()*

**Pregunta 2.**

Puntos: 10

2. Varias respuestas correctas: Ficheros - Pregunta 2: Indica cual, o cuales, de las siguien...

| | |
|---|---|
| Pregunta | Indica cual, o cuales, de las siguientes respuestas son correctas: |
| Respuesta | ✓ Al cerrar el flujo que tenemos en una variable (objeto) de tipo *FileWriter*, también se cierra el flujo del objeto *BufferedWriter*, si tenemos uno asociado.<br>✓ Podemos crear una instancia (variable, objeto) de tipo *FileReader* sin necesidad de crear una de tipo *File*. Bastaría con pasar como parámetro un *String* indicando la ruta del fichero que queremos leer.<br>La gestión de excepciones nos ayuda a mejorar el acceso a los ficheros.<br>No hay diferencias entre utilizar un *FileWriter* y un *BufferedWriter* desde el punto de vista de la eficacia. |
| Comentario correcto | Muy bien, sigue así!<br><br>Igualmente, te indico la explicación a las respuestas:<br><br>Al cerrar el flujo que tenemos en una variable (objeto) de tipo *FileWriter*, también se cierra el flujo del objeto *BufferedWriter*, si tenemos uno asociado.<br><br>CORRECTO: Se hizo un ejemplo en un video en el que al cerrar el FileWriter también cierra el BufferedWriter asociado ya que al cerrar el primero se cierra también el flujo del segundo. |

Podemos crear una instancia (variable, objeto) de tipo *FileReader* sin necesidad de crear una de tipo *File*. Bastaría con pasar como parámetro un *String* indicando la ruta del fichero que queremos leer.

CORRECTO: Sí. Existe un constructor que nos permite crear un FileReader.

La gestión de excepciones nos ayuda a mejorar el acceso a los ficheros.

INCORRECTO: Nos permite mejorar la gestión de errores pero no el acceso a los ficheros.

No hay diferencias entre utilizar un *FileWriter* y un *BufferedWriter* desde el punto de vista de la eficacia.

INCORRECTO: Sí las hay. El segundo es más eficiente en términos de escritura de muchos datos ya que no abre y cierra continuamente el fichero, sino que almacena la información en una pequeña memoria llamada Buffer y cuando está llena es cuando escribe en el fichero, optimizando la escritura en el archivo.

Comentario incorrecto

Al cerrar el flujo que tenemos en una variable (objeto) de tipo *FileWriter*, también se cierra el flujo del objeto *BufferedWriter,* si tenemos uno asociado.

CORRECTO: Se hizo un ejemplo en un video en el que al cerrar el FileWriter también cierra el BufferedWriter asociado ya que al cerrar el primero se cierra también el flujo del segundo.

Podemos crear una instancia (variable, objeto) de tipo *FileReader* sin necesidad de crear una de tipo *File*. Bastaría con pasar como parámetro un *String* indicando la ruta del fichero que queremos leer.

CORRECTO: Sí. Existe un constructor que nos permite crear un FileReader.

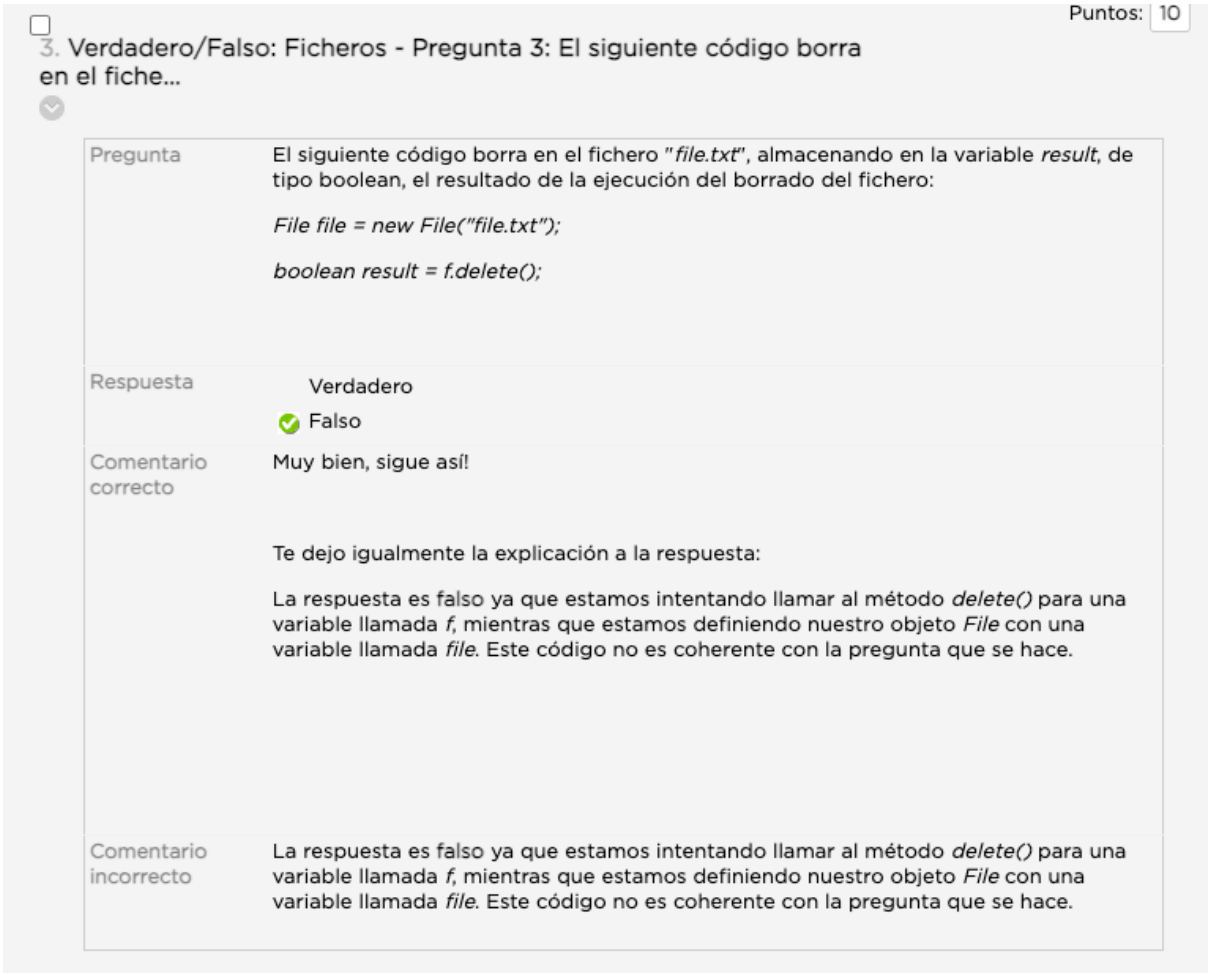
La gestión de excepciones nos ayuda a mejorar el acceso a los ficheros.

INCORRECTO: Nos permite mejorar la gestión de errores pero no el acceso a los ficheros.

No hay diferencias entre utilizar un *FileWriter* y un *BufferedWriter* desde el punto de vista de la eficacia.

INCORRECTO: Sí las hay. El segundo es más eficiente en términos de escritura de muchos datos ya que no abre y cierra continuamente el fichero, sino que almacena la información en una pequeña memoria llamada Buffer y cuando está llena es cuando escribe en el fichero, optimizando la escritura en el archivo.

**Pregunta 3.**



Puntos: 10

3. Verdadero/Falso: Ficheros - Pregunta 3: El siguiente código borra en el fiche...

| | |
|---|---|
| Pregunta | El siguiente código borra en el fichero "*file.txt*", almacenando en la variable *result*, de tipo boolean, el resultado de la ejecución del borrado del fichero:<br><br>*File file = new File("file.txt");*<br><br>*boolean result = f.delete();* |
| Respuesta | Verdadero<br>Falso |
| Comentario correcto | Muy bien, sigue así!<br><br>Te dejo igualmente la explicación a la respuesta:<br><br>La respuesta es falso ya que estamos intentando llamar al método *delete()* para una variable llamada *f*, mientras que estamos definiendo nuestro objeto *File* con una variable llamada *file*. Este código no es coherente con la pregunta que se hace. |
| Comentario incorrecto | La respuesta es falso ya que estamos intentando llamar al método *delete()* para una variable llamada *f*, mientras que estamos definiendo nuestro objeto *File* con una variable llamada *file*. Este código no es coherente con la pregunta que se hace. |

**Pregunta 4.**

Puntos: 10

4. Rellenar el espacio en blanco: Ficheros - Pregunta 4: Dado el siguiente código, indica qué ...

| | |
|---|---|
| Pregunta | Dado el siguiente código, indica qué debe completarse en el espacio que aparece con asteriscos (*) para que este trozo de código compile y escriba en un fichero que se pasa en el parámero *ruta*.<br><br>*FileWriter fw = new FileWriter(ruta);*<br><br>*BufferedWriter bw = new BufferedWriter(*******);*<br><br>*bw.write(cadena);* |

| Método de evaluación | Respuesta | Distingue entre mayúsculas y minúsculas |
|---|---|---|
| *Correspondencia exacta* | fw | Distingue entre mayúsculas y minúsculas |

| | |
|---|---|
| Comentario correcto | Muy bien, sigue así!<br><br>Te dejo igualmente la explicación a la pregunta:<br><br>Para mantener la coherencia con el código que se muestra, la respuesta debe ser: *fw*<br><br>*fw* es el nombre de nuestra instancia (variable, objeto) de tipo FileWriter.<br><br>Para crear un objeto de tipo BufferedWriter necesitamos una instancia de tipo *FileWriter*, que en este caso es *fw*.<br><br>*FileWriter fw = new FileWriter(ruta);*<br><br>*BufferedWriter bw = new BufferedWriter(fw);*<br><br>*bw.write(cadena);* |

**Pregunta 5.**

Puntos: 10

5. Varias respuestas correctas: Ficheros - Pregunta 5: Indica cuál de las siguientes pregunt...

| | |
|---|---|
| Pregunta | Indica cuál de las siguientes preguntas es incorrecta: |
| Respuesta | El método/función *readLine()* de la clase BufferedReader obtiene una línea de texto del flujo que está leyendo. Si llega al EOF, retorna null. |
| | Es posible capturar diferentes tipos de excepciones dentro del conjunto de instrucciones que hay en un elemento *try* |
| | El método *printStackTrace()* imprime en consola la pila de errores producidos por una excepción. |
| | ✓ No es posible acceder a una posición determinada de un fichero binario. |
| Comentario correcto | Muy bien, sigue así!<br><br>Te dejo igualmente la explicación a esta pregunta y sus respuesta:<br><br>El método/función *readLine()* de la clase BufferedReader obtiene una línea de texto del flujo que está leyendo. Si llega al EOF, retorna null. CORRECTO: Según la documentación Javadoc de este método, la función readLine obtiene una línea de texto. Si llega al End Of File (EOF), retorna null.<br><br>Es posible capturar diferentes tipos de excepciones dentro del conjunto de instrucciones que hay en un elemento *try*. CORRECTO: Sí, es posible poner varios *catch* para una misma sección *try*.<br><br>*El método printStackTrace() imprime en consola la pila de errores producidos por una excepción.* CORRECTO: Imprime la "pila" que es como se denomina al conjunto de líneas que llevan hasta la línea que produce la excepción.<br><br>*No es posible acceder a una posición determinada de un fichero binario.* INCORRECTO: Sí es posible acceder utilizando la clase *RandomAccessFile*. |

## Pregunta 6.

6. Verdadero/Falso: Ficheros - Pregunta 6: La sección finally de un bloque try-c...

| | |
|---|---|
| Pregunta | La sección *finally* de un bloque *try-catch* se ejecuta siempre y cuando se produzca una excepción dentro del bloque *try*.<br><br>} |
| Respuesta | Verdadero<br>✓ Falso |
| Comentario correcto | Muy bien, sigue así!<br><br>Igualmente te dejo la explicación a la pregunta:<br><br>Un bloque *finally* se ejecuta SIEMPRE. Se produzca excepción o no se produzca. |
| Comentario incorrecto | Falso.<br><br>Un bloque *finally* se ejecuta SIEMPRE. Se produzca excepción o no se produzca. |

## Pregunta 7.

Puntos: 10

7. Varias respuestas correctas: Ficheros - Pregunta 7: ¿Cuál o cuales de las siguientes afir...

| | |
|---|---|
| Pregunta | ¿Cuál o cuales de las siguientes afirmaciones son ciertas? |
| Respuesta | ✓ Podemos utilizar la clase *PrintWriter* para escribir datos en un fichero.<br>✓ Podemos crear una instancia (variable, objeto) de tipo *File* a partir de otra instancia de tipo *File* y un nombre en forma de cadena.<br>Con los flujos conseguimos abstraernos de la fuente de datos de origen, pero no de la de destino.<br>Los archivos se identifican en todos los sistemas operativos por el patrón: *[nombre].[extension]* |

| | |
|---|---|
| Comentario incorrecto | Podemos utilizar la clase *PrintWriter* para escribir datos en un fichero.<br><br>CORRECTO: En lugar de usar BufferedWriter podemos utilizar también la clase PrintWriter como hicimos en el primer ejemplo de escritura en ficheros de textos. |
| | Podemos crear una instancia (variable, objeto) de tipo *File* a partir de otra instancia de tipo *File* y un nombre en forma de cadena.<br><br>CORRECTO: Existe un constructor (ver pdf de teoría) que nos indica que sí podemos crear una instancia de este tipo a partir de dichos parámetros. |
| | Con los flujos conseguimos abstraernos de la fuente de datos de origen, pero no de la de destino.<br><br>INCORRECTO: Con los flujos, nos abstraemos de las fuentes de origen y de destino. De ambas. |
| | Los archivos se identifican en todos los sistemas operativos por el patrón: *[nombre].[extension]*<br><br>INCORRECTO: En algunos sistemas operativos (Por ejemplo los basados en Linux y UNIX) no hace falta extensión. |

**Pregunta 8.**

Puntos: 10

8. Verdadero/Falso: Ficheros - Pregunta 8: Los dos extractos de código siguiente...

Pregunta

Los dos extractos de código siguientes hacen lo mismo:

```
linea = br.readLine();
while (linea !=null) {
  totalLineas = totalLineas+"\n"+linea;
  linea = br.readLine();
}
```

```
while ((linea=br.readLine()) !=null) {
  totalLineas = totalLineas+"\n"+linea;
  linea = br.readLine();
}
```

Respuesta

Verdadero

Falso

Comentario incorrecto

La respuesta es falso.

En el primer código se hace de forma separada: primero se asigna a línea la lectura del flujo y luego se compara con *null* por si se ha llegado al final del fichero.

En el segundo código todo se hace dentro de la condición del *while*. Se asigna primero a linea el resultado de la llamada a *readLine()* (fíjate en los paréntesis que prevalecen) y luego se compara con *null*.

Pero se lee dos veces la línea (en el cuerpo del while y en el test del while).

**Pregunta 9.**

Puntos: 1,0

## 9. Verdadero/Falso: Ficheros - Pregunta 9: Tras la ejecución de la siguiente fun...

| | |
|---|---|
| Pregunta | Tras la ejecución de la siguiente función con los parámetros: *"file.txt"*, *"Hola"* y *true*, la función retorna: *"Hola+\n"*<br><br>*public static String escribirFicheroTexto(String rutaFichero, String cadena, boolean tipo) {*<br>*File fichero = null;*<br>*FileWriter fw = null;*<br>*BufferedWriter bw = null;*<br>*String tmp;*<br>*try {*<br>*fichero = new File(rutaFichero);*<br>*fw = new FileWriter(fichero, tipo);*<br>*bw = new BufferedWriter(fw);*<br>*}*<br>*catch(IOException ioe) {*<br>*System.out.println("Excepción al crear el FileWriter");*<br>*ioe.printStackTrace();*<br>*}*<br>*try {*<br>*bw.write(cadena);*<br>*tmp = cadena;*<br>*bw.write("\n");*<br>*tmp += "\n";*<br>*} catch (IOException e1) {*<br>*e1.printStackTrace();*<br>*}*<br><br>*try {*<br>*bw.close();*<br>*fw.close();*<br>*} catch (IOException e) {*<br>*e.printStackTrace();*<br>*}*<br>*return tmp;*<br>*}* |

| | |
|---|---|
| Respuesta | Verdadero<br>✓ Falso |
| Comentario correcto | Muy bien, ánimo y sigue así!<br><br>Te dejo la explicación a la respuesta:<br><br>La respuesta es falso:<br><br>La función retorna "*Hola*", que es diferente a la cadena "Hola+\n".<br><br>La secuencia de escape *"\n"* genera un salto de línea en el fichero, de forma que la siguiente vez que se escriba, lo hará en la siguiente línea. Pero no muestra el caracter +. |
| Comentario incorrecto | La respuesta es falso:<br><br>La respuesta es falso:<br><br>La función retorna "*Hola*", que es diferente a la cadena "Hola+\n".<br><br>La secuencia de escape *"\n"* genera un salto de línea en el fichero, de forma que la siguiente vez que se escriba, lo hará en la siguiente línea. Pero no muestra el caracter +. |